UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERIA SUPERIOR DE TELECOMUNICACIÓN

PROYECTO FINAL DE CARRERA

CRUCIGRAMA PARA AULA LÚDICA EN ENTORNOS 3D

AUTOR: LUCÍA PAYO MOLINA
TUTOR: MARÍA BLANCA IBÁÑEZ

20 de Septiembre de 2012

Resúmen

En este proyecto se desarrolla una aplicación que permite la gestión y el uso de crucigramas como objetos de aprendizaje en líneas tri-dimensionales. Los crucigramas son utilizados como herramientas que permiten a los estudiantes reforzar los conocimientos sobre un determinado tópico en un ambiente de aprendizaje inmersivo.

La aplicación consta de dos partes. La primera es el plug-in que permite utilizar los crucigramas dentro del entorno virtual tri-dimensional que sirve de base a este desarrollo: OpenSim. La segunda es la aplicación web que permite la creación y la gestión del contenido de los crucigramas: OpenWeb.

Los estudiantes refuerzan sus conocimientos generando un crucigrama sobre un tópico particular y con determinado grado de dificultad desde el mundo virtual 3D. El estudiante puede resolver el crucigrama y tiene algunas herramientas a su disposición, como por ejemplo pistas. Una vez finalizado el crucigrama se recogen una serie de datos, la nota obtenida en cada una de las preguntas, el número de pistas utilizadas o la nota final del crucigrama, y se almacenan en una base de datos que eventualmente podrá ser consultada haciendo uso de OpenWeb.

OpenWeb es una aplicación web que es utilizada tanto por profesores como por estudiantes. Para el profesor constituye una herramienta de administración y seguimiento del desempeño de los alumnos. La aplicación le permite además, añadir preguntas o modificar las ya existentes. Para el estudiante se trata de una herramienta de apoyo y motivación, donde puede consultar los resultados obtenidos por él/ella o sus compañeros y además, puede proponer nuevas preguntas para que previa aprobación del profesor, sean incluidas en la batería de preguntas ya existente.

Índice

1. Introducción

En este primer capítulo se explicarán brevemente los conceptos de e-learning y mundos virtuales colaborativos tridimensionales. Además, se dará una justificación para combinar estos dos conceptos y concretamente la motivación que ha llevado a desarrollar el presente proyecto fin de carrera.

1.1. E-learning

Se denomina e-learning [1] o aprendizaje electrónico a la educación a distancia a través de canales electrónicos donde los alumnos pueden comunicarse y colaborar entre ellos. Los profesores pueden utilizar elementos multimedia como imágenes, vídeos o animaciones para enseñar un concepto, sin embargo es posible también que el alumno participe de forma activa en su educación a través de juegos, mundos virtuales tridimensionales, foros, etc...

Una de las ventajas más evidentes del e-learning [2] es que elimina las barreras espacio-temporales dando una gran flexibilidad al alumno para compaginar los estudios con cualquier otra actividad, sin embargo no es la única, el aprendizaje electrónico también permite una gestión real del conocimiento propiciando un entorno colaborativo en el que todos alumnos pueden participar intercambiando ideas u opiniones, i.e. generando conocímiento. Además favorece la comunicación alumno-profesor a través de canales asíncronos o síncronos, y la actualización inmediata de contenido por parte del profesor.

Pero en el e-learning no todo son ventajas, también existen algunos inconvenientes que hacen que el aprendizaje electrónico no sea un método suficiente por sí mismo, sino más bien una herramienta que completa la enseñanza presencial tradicional. Tanto alumnos como profesores deben tener unos conocimientos mínimos del canal de distribución, ya sea internet, intranet o un mundo virtual 3D. Además exige un mayor trabajo por parte del profesor y los alumnos deben tener capacidades de aprendizaje autónomo.

El crecimiento del e-learning [3] es innegable según el informe presentado por la Dirección General de Educación y Cultura de la Comisión Europea. Únicamente el 15 % de las universidades europeas no hacen uso del aprendizaje electrónico y el 64 % de las universidades ha mostrado su voluntad en ampliar la oferta de

e-learning. Estos datos nos confirman la importancia que está adquiriendo el aprendizaje electrónico y la necesidad de investigar e innovar en este campo.

1.2. Mundos Virtuales

Los mundos virtuales colaborativos tridimensionales son comunidades en línea que simulan un entorno real o imaginario en los cuales los usuarios pueden interactuar entre sí a través de sus avatares y usar objetos o bienes virtuales. Existen muchas aplicaciones para la realidad virtual, entre ellas se puede destacar el desarrollo de simuladores que ayudan a la formación de diversas profesiones, como puede ser en medicina (cirugía) o en la formación de pilotos, también se utiliza para simular monumentos y edificios lo cual puede facilitar el trabajo a profesionales como los arquitectos, otra de las aplicaciones mas extendidas está en el campo del ocio con el desarrollo de videojuegos y por supuesto en educación, donde se pueden construir herramientas interactivas que complementen la formación de los alumnos aportando una experiencia educativa diferente.

Para el desarrollo de este trabajo hemos elegido una plataforma de código abierto para la creación de mundos virtuales colaborativos tri-dimensionales llamada OpenSim[4] que proviene de otro mucho más conocido, pero de código propietario, llamada Second Life[5]. En OpenSim, al igual que en Second Life, tanto objetos como avatares están representados típicamente como modelos 3D. El avatar es la representación del jugador (en nuestro caso el estudiante) en el mundo y la vía a través de la cual puede interactuar y relacionarse con el resto de jugadores. De esta manera el avatar puede personalizarse, pudiendo elegir una serie de variables que modificarán su aspecto. Por otro lado están los objetos, OpenSim posee un modo construcción en el cual cualquier jugador puede crear cosas partiendo de formas básicas o *prims*[1]. Además puede darle funcionalidad a través de guiones (*scripts*) que definen el comportamiento del objeto ante ciertos eventos como puede ser tocarlo, hablarle o sentarse sobre él. El lenguaje que se utiliza para escribir los *scripts* es el *Linden Script Language*[6] desarrollado por Second Life. Gracias a estas herramientas OpenSim permite a sus usuarios construir mundos y entornos que proporcionen nuevas experiencias al resto de jugadores.

Al igual que muchos otros mundos virtuales similares, OpenSim es persistente en el tiempo gracias a una arquitectura cliente-servidor. De este modo no se pierden los avances y puede conectarse desde cualquier lugar.

[1] *Prim* es el nombre que se le da en OpenSim a la unidad básica de construcción

1.3. Objetivos

E-learning y mundos virtuales son dos conceptos que pueden funcionar muy bien combinados. Es posible recrear un entorno educativo dentro de un mundo virtual tridimensional que permita al alumno desarrollar sus capacidades en cierta materia de una manera diferente. El presente proyecto fin de carrera consta de dos partes bien diferenciadas:

- Por un lado está la construcción de un crucigrama interactivo en el mundo virtual tridimensional donde el alumno puede elegir el tema y la dificultad para su generación. Una vez resuelto, el alumno dará por finalizado el crucigrama obteniendo una puntuación.
- Por otro lado se dispone de una página web para alumnos y profesores con las siguientes funcionalidades:
  - Los alumnos podrán ver su progreso, consultando la puntuación obtenida en los crucigramas según tema y dificultad.
  - Existen rankings para motivar a los alumnos a repetir los crucigramas e intentar conseguir una mejor puntuación.
  - Los alumnos podrán proponer preguntas al profesor para que sean añadidas al crucigrama.
  - Se podrán encontrar pistas o apuntes acerca de las preguntas que ayuden al alumno a encontrar la información que necesita.
  - El profesor podrá hacer un seguimiento de cada alumno.
  - El profesor podrá consultar la media de aciertos de cada pregunta pudiendo así modificarla si lo cree preciso.
  - El profesor dispone de una interfaz para añadir nuevas preguntas al crucigrama, borrarlas o aceptar propuestas de los alumnos.

Con estos elementos el alumno puede medir sus conocimientos en la materia resolviendo crucigramas y encontrar cierta motivación a través de la página web. El profesor puede controlar el progreso de todos los alumnos y utilizar esta herramienta para motivar en otros campos a los alumnos, por ejemplo ofreciendo puntos extra al alumno que más propuestas haya hecho o al que vaya primero en el ranking.

2. Crucigrama

2.1. Descripción de las tareas

Como se ha mencionado anteriormente el proyecto se divide en dos partes y una de ellas es el diseño e implementación del crucigrama en OpenSim. Para poder realizar el propósito marcado se definieron las siguientes tareas generales:

1. Construcción de objetos en OpenSim: Manejar la herramienta de construcción de OpenSim y construir los objetos necesarios para la interacción con el usuario.
2. Añadir funcionalidad a los objetos en OpenSim: Creación de los *scripts* necesarios en los objetos para permitir la interacción con el crucigrama.
3. Generación del crucigrama: Dado un número acotado de palabras, en nuestro caso elegimos 10 palabras, ser capaces de construir un crucigrama correctamente.
4. Generación de las bases de datos: Esta tarea incluye tanto el diseño de las bases de datos para la persistencia como el desarrollo de los programas necesarios para tener acceso a los datos.
5. Comunicación entre OpenSim y el servidor: Desarrollo de los programas que actuarán como mediadores entre OpenSim, la base de datos y la generación del crucigrama.

2.2. Herramientas y tecnologías utilizadas

Para alcanzar la funcionalidad completa es necesario decidir cómo se van a desarrollar las tareas anteriores y qué herramientas o tecnologías se van a utilizar para llevarlas a cabo. Esta sección nos va a ayudar a definir en detalle más adelante la funcionalidad concreta de cada uno de los puntos.

2.2.1. Generación del crucigrama

La generación de un crucigrama no es una tarea trivial y precisa de un algoritmo complejo. En el presente proyecto no se pretende desarrollar un algoritmo generador de crucigramas y por ello se ha utilizado uno ya existente llamado *php crossword generator*[2]. El código es de libre distribución y se puede descargar en la página indicada a pie de página.

Cuando el alumno elija tema y dificultad para su crucigrama se deberán escoger diez palabras al azar que cumplan los requisitos impuestos de entre las posibles. A partír de estas diez palabras la aplicación construirá un crucigrama reutilizando gran parte del código de *php crossword generator* y adaptándolo al presente proyecto. Se ha decidido que los crucigramas sean de diez palabras porque es el máximo que acepta *php crossword generator*, además un crucigrama más grande aumentaría el tiempo de generación.

El generador de crucigramas elegido es muy conveniente para el propósito perseguido en este proyecto porque genera crucigramas basándose en una matriz 15x15, lo que permite construir una estructura estática dentro del mundo virtual tridimensional. Las palabras ocupan posiciones dentro de esa matriz y la aplicación calcula la posición inicial de la palabra en coordenadas *x*-*y* y si es vertical u horizontal. Con estos datos junto con la longitud de la palabra se puede posicionar fácilmente dentro de la matriz.

El algoritmo utilizado usa la técnica de backtracking para hacer los cruces entre palabras hasta llegar al crucigrama final. La idea del backtracking se asemeja a un recorrido en profundidad en un grafo dirigido como el de la figura 1 [7], cuya finalidad es encontrar la solución para algún problema construyendo soluciones parciales a medida que progresa el recorrido del grafo. Estas soluciones parciales limitan las regiones en las que se puede encontrar la solución completa, de esta manera cuando el algoritmo llega a un punto muerto lo que hace es volver hacia atrás y repetir el proceso por otro camino diferente del grafo, buscando así la solución completa en otra región.

[2]http://phpcrossword.sourceforge.net/

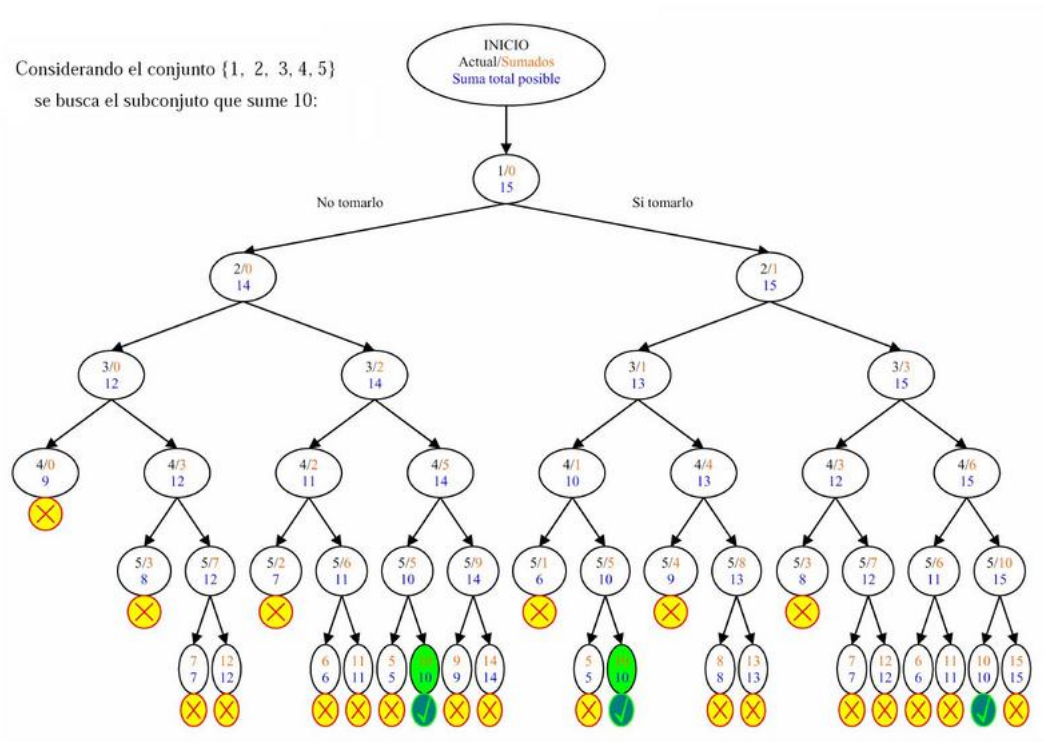


Figura 1: Grafo dirigido para backtraking

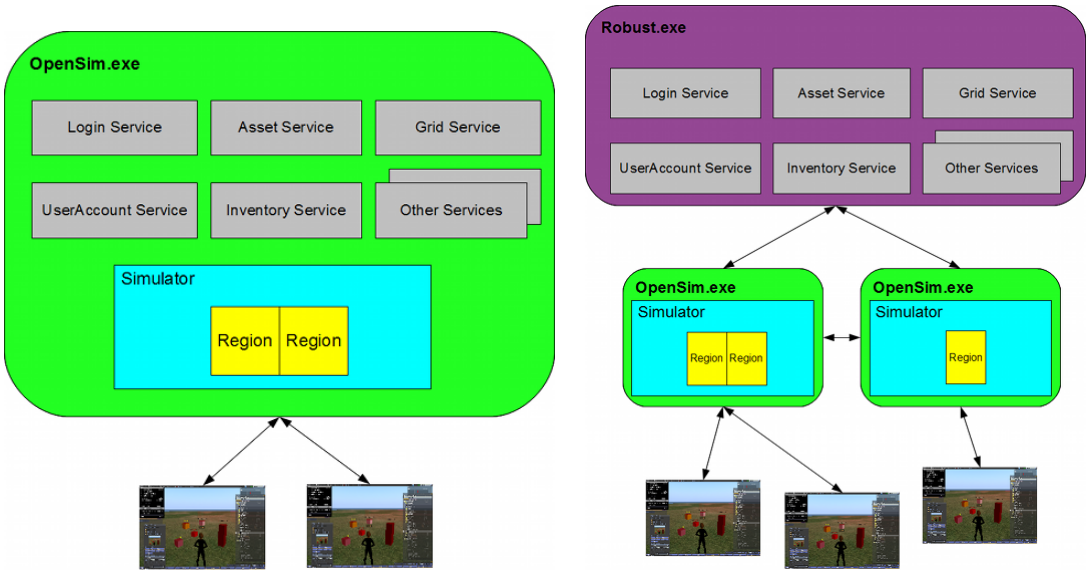
2.2.2. OpenSim

OpenSim [8] es un servidor multi plataforma y multi usuario 3D en código abierto. Permite crear entornos o mundos virtuales a los que se puede acceder a través de multitud de clientes y protocolos distintos. OpenSim está escrito en lenguaje C y puede ejecutarse tanto en Windows como en sistemas Unix.

El mundo en OpenSim está dividido en porciones de terreno cuadradas llamadas *sims* o regiones de tamaño 256x256[3] que a su vez forman parte de una matriz mayor llamada rejilla (*grid*) de tamaño máximo 65536x65536. Al crear la región en el servidor hay que indicarle en qué posición debe colocarse dentro del *grid*, una vez creada el resto de regiones no pueden ponerse en esa posición. En nuestro caso solo hemos creado una sola región en la posición 1000,1000 pero OpenSim permitiría ampliar el terreno colocando otras regiones adyacentes a la ya existente.

[3]La unidad de medida de las regiones es el tamaño de un *prim* recién construido

Las regiones que forman un *grid* pueden estar alojadas todas en el mismo servidor o en servidores diferentes dando lugar a dos modos de funcionamiento. En el primer caso toda la información y todos los servicios necesarios para el funcionamiento del *grid* están en la misma máquina que aloja las regiones, por lo tanto la configuración de este modo es más sencilla y no requiere de módulos adicionales, a esta configuración se le llama *stand-alone*. Si por el contrario tenemos regiones en diferentes máquinas se necesita un módulo que controle la información y los servicios en todas las regiones, en la figura 2 quedan representados los dos modos de funcionamiento [9].



(a) Regiones en la misma máquina (b) Regiones en diferentes máquinas

Figura 2: Modos de funcionamiento de OpenSim

Una vez que están creadas las regiones hay que dar de alta a los usuarios, esto se puede hacer a través de comandos directamente desde el servidor o desde una página web que se comunique con el servidor de OpenSim. Existen varios clientes para conectarse a OpenSim (Hippo, Imprudente, Singularity, etc.), es necesario configurarlos para poderse conectar a un *grid* determinado, para ello hay que indicar la IP del servidor que contiene el *grid* seguido por el puerto 9000 que es donde se encuentra escuchando por defecto OpenSim. Cada región también ocupa un puerto, por defecto se le asignan puertos consecutivos a partir del 9001 pero esto es transparente para el cliente y no es necesario configurar nada. Toda la configuración viene detallada en el Anexo A.

En el presente proyecto fin de carrera la configuración utilizada ha sido de

stand-alone con los puertos definidos por defecto y con una sola región. Como cliente se ha elegido Imprudence porque tiene versiones tanto para Windows como para sistemas Unix y es actualizado habitualmente.

2.2.3. Construcción de objetos en OpenSim

OpenSim ofrece una herramienta para la construcción de objetos tridimensionales. Partiendo de formas básicas como pueden ser cubos, esferas o pirámides se pueden moldear hasta conseguir la forma deseada del objeto a crear. A las unidades básicas, ya estén alteradas o en su forma original, se les llama *prims*. El constructor de OpenSim ofrece la posibilidad de juntar varios *prims* para formar un solo objeto, de esta manera se pueden conseguir formas más complejas.

Con esta herramienta hemos construido los elementos necesarios para la visualización del crucigrama. Por un lado existe una matriz de 15x15 donde se generará el crucigrama, esta matriz está unida a una mesa con botones que ofrecen diferentes funcionalidades al usuario. Por otro lado hay un panel en el que se puede elegir la dificultad para el crucigrama, y por último hay una caja que contiene pequeños cubos que representan cada uno de los temas. En la figura 3 se pueden apreciar todos los objetos creados.

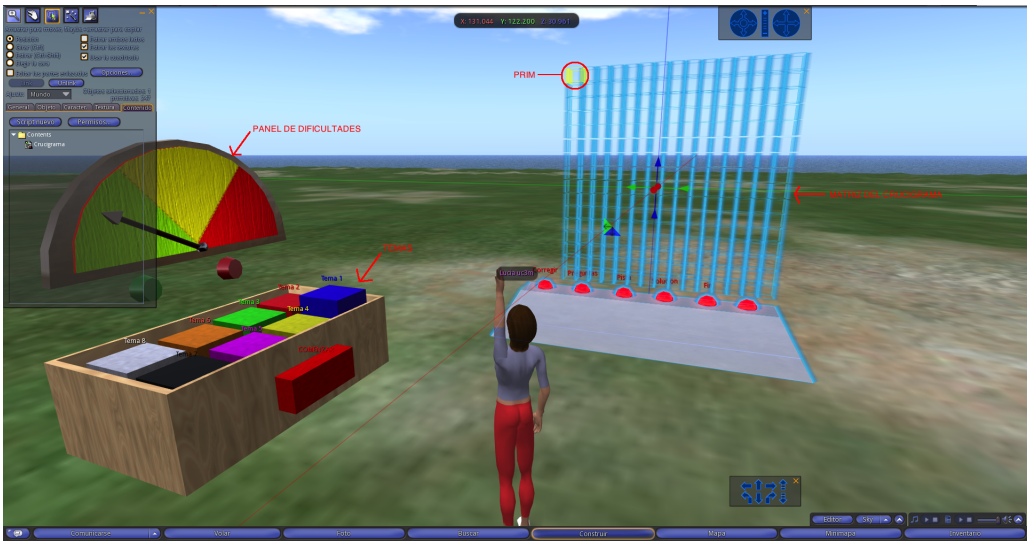


Figura 3: Elementos construidos

OpenSim permite importar texturas, imágenes y modelos 3D. En este proyecto

se han importado varias imágenes para poder escribir en el crucigrama, concretamente todos los números y letras. No se han importado modelos 3D pero sería posible crear un crucigrama visualmente más atractivo con una herramienta de modelado, como por ejemplo Google Sketchup o Maya.

2.2.4. Generación de las bases de datos

Es necesario que exista una persistencia de los datos, tanto para almacenar todas las preguntas y palabras que pueden aparecer en los crucigramas como para guardar los resultados obtenidos por los alumnos. Para ello se necesita una base de datos y en este caso hemos elegido MySQL.

MySQL[10] es un sistema gestor de bases de datos de licencia libre GPL, con gran prestigio a nivel mundial, desarrollado por Oracle. Escrito en C y C++, cuenta con múltiples funcionalidades y características que lo convierten en una interesante alternativa al gestor principal del mismo nombre que la empresa, Oracle.

Para este proyecto, se utiliza MySQL Community Server en su versión 5.1.41

2.2.5. Añadir funcionalidad a los objetos en OpenSim

OpenSim no solo ofrece una herramienta para la construcción de nuevos objetos, sino que también permite darles funcionalidad. Esta tarea se consigue a través de *scripts* que se asocian a cada objeto. Los *scripts* están escritos en un lenguaje de programación llamado *Linden script Language* (LSL) basado en Java y C.

LSL es un lenguaje basado en estados y eventos que se disparan cuando algún avatar interactúa con un objeto o dos objetos interactúan entre sí. El evento puede ser capturado por el *script* del objeto para realizar cualquier acción. Los eventos vienen predefinidos y el usuario no puede crearlos, por lo tanto es necesario consultar el API de LSL[4] para saber que tipo de eventos puede manejar OpenSim. Dentro del *script* se puede cambiar de estado asociado a un objeto dependiendo del evento que se capturó, para así hacer una programación más clara, pero no es estrictamente necesario utilizar más de un estado, que por defecto es el estado "default" y hay que definirlo obligatoriamente.

[4]http://wiki.secondlife.com/wiki

Para explicar mejor el funcionamiento general de LSL vamos a apoyarnos en el ejemplo[5] de la figura 4. Lo que se pretende con este *script* es simular el funcionamiento de una bombilla, por lo que es lógico pensar en dos estados, uno cuando está encendida y otro cuando está apagada. El estado "default" es obligatorio en todos los *scripts* y en este caso representa a la bombilla encendida. Por otro lado está el estado "off" que ha sido definido por el usuario y representa a la bombilla apagada. Dentro de cada estado se definen los eventos que se quieren capturar, en este caso en los dos estados se capturan los mismos eventos, "state_entry" y "touch_start". El evento "state_entry" se lanza nada más entrar en el estado, es lo primero que se ejecuta al cambiar de un estado a otro y como se observa en ambos casos el objeto dice al mundo si se enciende o si se apaga además de cambiar su color. El segundo evento se lanza cuando un avatar toca el objeto y la acción que se realiza es simplemente cambiar de estado, así cuando el avatar toque el objeto bombilla, si ésta se encuentra en estado "default" (encendida) pasará al estado "off" (apagada).

```
default //El estado default es obligatorio
{
    state_entry() // se ejecuta cada vez que entra en el estado
    {
        llSay(0, "Me enciendo!"); // el objeto habla
        llSetColor(<1.0, 1.0, 1.0>, ALL_SIDES); // Configura todos los lados del objeto para que sean brillantes
        // Es necesario poner ';' al final de cada linea (no poner ';' al final de un estado)
    }

    touch_start(integer total_number) // Evento que aparece cuando un avatar toca el objeto
    {
        state off; // manda al script al otro estado, en concreto al estado "off" (apagado)
    }
} // Este corchete indica el final del estado "default"

state off // Otro estado a parte de "default". Este no es obligatorio.
{
    state_entry() // Se ejecuta cuando se entra en el estado
    {
        llSay(0, "Me apago!"); // De nuevo el objeto habla
        llSetColor(<0.0, 0.0, 0.0>, ALL_SIDES); // Se configuran los lados del objeto para que sean oscuros
    }

    touch_start(integer total_number)
    {
        state default;
    }
}
```

Figura 4: Ejemplo sencillo de un *script* para una bombilla

Existen un gran número de funciones definidas que se pueden consultar en el api de LSL, en el ejemplo anterior se utilizaron *llSay* y *llSetColor*. El desarrollador también puede definir sus propias funciones fuera de los estados para llamarlas dentro de los eventos.

[5]Fuente: http://wiki.secondlife.com/wiki/Help:Getting_started_with_LSLes

Los *scripts* se pueden comunicar entre sí gracias al paso de mensajes. Se pueden capturar utilizando el evento *listen(integer channel, string name, key id, string message)*. OpenSim tiene muchos canales que pueden ser utilizados por los *scripts* pero algunos están reservados, como el PUBLIC_CHANNEL alojado en el canal 0 que es visible para todos los avatares.

Otro punto interesante y a tener en cuenta a la hora de desarrollar *scripts* en OpenSim es que el sistema asigna un id de tipo *key* a todos los elementos en el mundo (objetos, avatares, texturas, imágenes, etc.). Esto no es más que una manera de identificar inequívocamente cada elemento.

LSL se interpreta y ejecuta en el servidor y no en el cliente. Aunque el editor de *scripts* es parte del visor de OpenSim, el propio *script* se ejecuta en el servidor, que envía los resultados a través de la red al visor, donde se pueden ver.

2.2.6. Comunicaciones

Para que todo el sistema funcione es necesario que OpenSim sea capaz de comunicarse tanto con el programa generador de crucigramas como con la base de datos para almacenar los resultados de los crucigramas. PHP es el lenguaje elegido para el desarrollo de los programas encargados de la comunicación ya que es fácil de integrar con *php crossword generator*, permite el manejo de bases de datos MySQL y puede recibir peticiones http externas comportándose como un servicio web.

2.3. Arquitectura

Este apartado pretende clarificar los bloques que componen el sistema del crucigrama y la interacción que existe entre ellos. Comenzaremos enumerando los diferentes elementos en los que se divide el sistema:

- El servicio web implementado en PHP que se encargará de comunicar todos los elementos entre si siendo el punto central del sistema. En el desarrollo del presente proyecto se ha alojado en un servidor Apache.
- El servidor de OpenSim donde se encontrarán los *scripts* que se comunican vía HTTP con el servicio web. También es el encargado de persistir todos

los elementos pertenecientes al mundo virtual y aceptar las conexiones de los alumnos.

- El cliente de OpenSim que se encontrará en todos los equipos desde los que se acceda a OpenSim y que permite la interacción con el entorno virtual.
- El generador de crucigramas, programa desarrollado en PHP y alojado en un servidor Apache.
- La base de datos MySQL que permite la persistencia de la información.

El alumno se conectará al servidor de OpenSim a través del cliente, pudiendo interactuar con el entorno para generar un nuevo crucigrama. Para ello será necesario obtener la información sobre las preguntas y las posiciones que van a ocupar las palabras dentro del crucigrama. El servidor de OpenSim se comunicará con el servicio web para obtener toda esta información. El servicio web se comunica a su vez tanto con la base de datos como con el programa que genera los crucigramas, por lo que obtiene las palabras y se las pasa a *php crossword generator*. Una vez finalizado el crucigrama, el servidor de OpenSim volverá a ponerse en contacto con el servicio web para mandarle los resultados obtenidos por el alumno, que serán almacenados en la base de datos.

En la figura 5 se puede ver un gráfico que representa todos los elementos junto con las comunicaciones necesarias para el sistema.

2.4. Funcionalidad

Una vez clarificada la arquitectura y tomadas las decisiones para poder realizar las tareas generales que se definieron anteriormente, se puede entrar en detalle a definir la funcionalidad concreta del sistema. En esta parte el único actor en la interacción con el crucigrama en el mundo virtual es el alumno.

El alumno cuando entra a OpenSim podrá interactuar con los objetos presentes en el entorno. Concretamente podrá elegir un tema interactuando con la caja que se muestra en la figura 3, cada tema queda representado por un cubo de un color diferente que se eleva sobre los demás cuando el usuario pincha sobre él. También podrá elegir una dificultad a través del panel, la flecha se moverá a izquierda y derecha cuando el usuario pulse uno de los dos botones, verde para bajar la dificultad y rojo para subirla. Posteriormente pulsando el botón comenzar que se

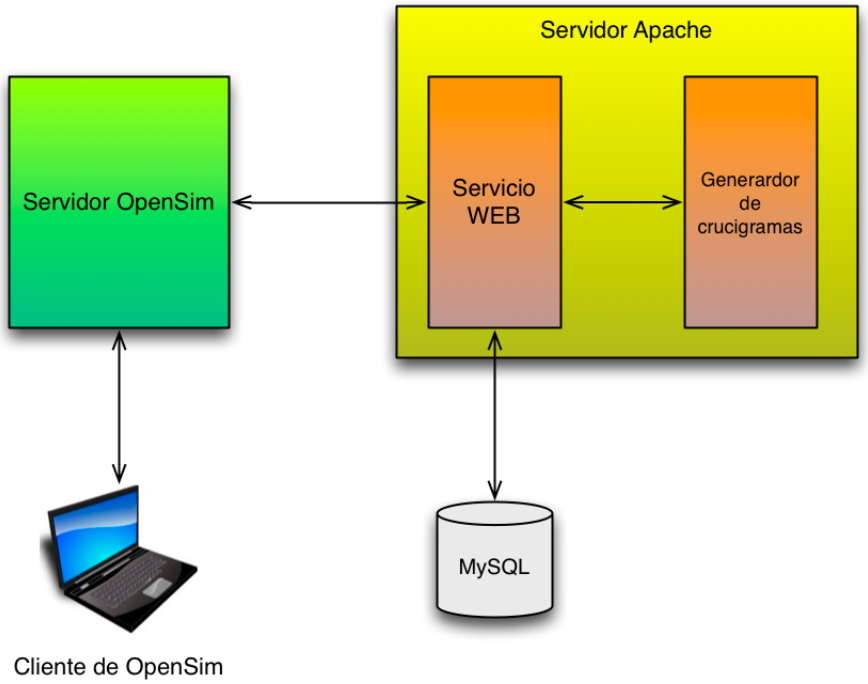


Figura 5: Arquitectura de la aplicación

encuentra en la caja de dificultades, se genera el crucigrama, formándose en el espacio delante del avatar. Se van colocando las casillas necesarias dependiendo de las preguntas que el sistema haya elegido de la base de datos. La interacción con el crucigrama se explicará al detalle más adelante en la sección 2.4.2, pero básicamente el alumno podrá escribir las palabras sobre las casillas interactuando con la matriz generada en el espacio.

Mientras el alumno resuelve el crucigrama existen otras funcionalidades útiles. Se pueden pedir pistas para una pregunta, lo cual desvelará determinadas letras de la solución. También se pueden volver a pedir las preguntas por si se da el caso de que el alumno cerrara el panel donde aparecen.

Finalmente, una vez resuelto el crucigrama el alumno pulsará el botón de corregir lo que le indicará que palabras están bien y la puntuación conseguida. Una vez hecho esto la puntuación se guardará en la base de datos y el alumno podrá pulsar el botón solución para ver las respuestas.

Por último el alumno puede utilizar el botón de fin para limpiar el crucigrama y

eliminar del mundo virtual el crucigrama generado dejando el sistema en el punto de partida.

En la figura 6 se sintetiza en un diagrama de casos de uso la funcionalidad anteriormente descrita.

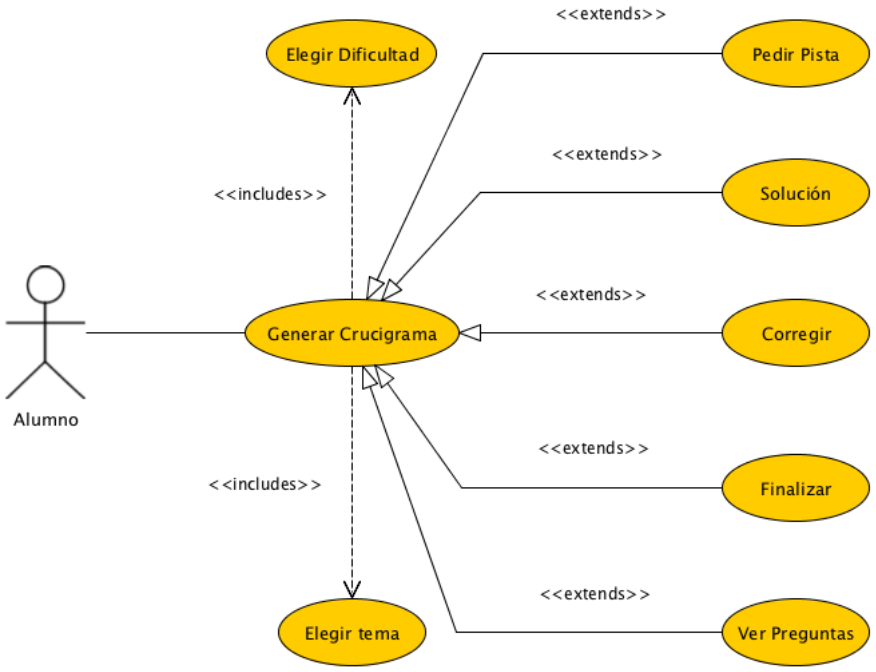


Figura 6: Casos de uso para el crucigrama

2.4.1. Generar crucigrama

En el mundo existen tres objetos: el panel de dificultad, la caja de temas y el crucigrama (ver figura 3). Para generar un crucigrama el alumno deberá elegir un tema y su dificultad. Para elegir una dificultad, el estudiante pulsará los botones verde y rojo que se encuentran debajo del panel. Al hacerlo la flecha del objeto panel (similar al panel de revoluciones de un automóvil) se moverá a la izquierda o a la derecha conmutando entre los tres niveles posibles representados por los colores verde, amarillo y rojo que se corresponden con los niveles de dificultad bajo, medio y alto respectivamente. También es necesario elegir un tema, para ello existe una caja con cubos de colores en su interior que representan los ocho temas posibles. Al hacer click sobre uno de los cubos este se elevará sobre los demás marcando

así el tema elegido. Una vez hecho esto el alumno pulsará el botón comenzar y el crucigrama empezará a generarse segundos después en el espacio. Una vez hayan aparecido todas las casillas que forman el crucigrama aparecerán las preguntas en un panel arriba a la derecha de la pantalla y ya estará todo listo para resolverlo.

Tras explicar el funcionamiento de la generación del crucigrama vamos a pasar a describir más en detalle como el sistema lleva a cabo esta tarea y los elementos que participan. Lo primero es definir cuantos *scripts* existen en OpenSim:

- *script* del panel de dificultad: Este *script* es el encargado de reunir la información sobre tema y dificultad elegida por el alumno y mandársela al *script* del crucigrama cuando el alumno pulsa el botón comenzar. Este *script* controla las acciones realizadas sobre los botones rojo y verde, como las acciones realizadas sobre el botón comenzar.
- *script* de la flecha: Se encarga de rotar el objeto para indicar la dificultad sobre el panel. Se mantiene escuchando en un canal las indicaciones del *script* del panel para moverse cuando el alumno pulsa los botones verde o rojo. Este *script* solo controla el objeto de la flecha.
- *script* de la caja de temas: Captura el tema elegido por el alumno y se lo comunica al panel. Controla los cubos que se encuentran dentro de la caja elevándolos cuando son elegidos.
- *script* del crucigrama: Es el *script* más complejo y el encargado de controlar todas las funciones además de recibir la información por parte del *script* del panel sobre el tema y dificultad elegidos por el alumno. Controla tanto la matriz como la mesa con los botones de las diferentes funciones.

Para la elección de la dificultad, cuando el alumno pulsa uno de los botones (rojo o verde), el *script* del panel captura el evento *touch_ start(integer num_ detected)*. El panel está formado por varios *prims*, en el *script* se controla que *prim* ha sido tocado gracias al parámetro *num_ detected* que identifica el número del *prim* que ha sido pulsado dentro del conjunto, de esta manera se puede distinguir cual de los dos botones ha sido utilizado por el alumno. La flecha tiene su propio *script* dentro del panel cuyo cometido es definir su posición utilizando coordenadas polares debido a que el movimiento es rotativo. El *script* de la flecha y el del panel se comunican entre sí, por eso cuando el alumno pulsa un botón controlado por el *script* del panel este informa al *script* de la flecha que debe moverse a través del canal 51. El evento *listen(integer channel, string name, key id, string message)* es el

encargado de recoger los mensajes que se lanzan. Gracias a los parámetros se pueden filtrar, con *integer channel* podemos filtrar los mensajes por canal, con *string name* se puede recoger el mensaje de un objeto o avatar determinado filtrando por su nombre, lo mismo que con *key id* solo que filtrando por el identificador, en *string message*, como es de esperar, se encuentra el mensaje enviado. Para mandar un mensaje simplemente hay que llamar a la función *llSay(integer channel, string message)*. En la figura 7 se encuentra el *script* que contiene la flecha del panel, el evento *state_ entry()* es capturado cuando se resetea el *script* y es donde se define el estado inicial de la flecha. Es aquí donde se indican los eventos que se van a capturar, en este caso como se van a capturar mensajes se activa el evento *listen* definiendo el canal.

```
//key panel ="b8ba0b15-7c31-45ad-85fb-f72043b7846e";
rotation medio = <-0.36384,-0.309014,0.682471,0.553502>;
rotation facil = <-0.672223,-0.545860,0.381836,0.323038>;
rotation dificil = <0.051802,0.018935,0.781985,0.620852>;

default
{
    state_entry()
    {
        llSetLinkTexture(LINK_SET,TEXTURE_TRANSPARENT,ALL_SIDES);
        llListen(51,"",NULL_KEY, "");
        llSetLocalRot(medio);
    }

    listen(integer channel, string name, key id, string message){
        //La flecha se hace visible
        if(message == "poner")
            llSetLinkTexture(LINK_SET,"00000000-0000-2222-3333-100000001036",ALL_SIDES);
        //La flecha se hace invisible
        if(message == "quitar")
            llSetLinkTexture(LINK_SET,TEXTURE_TRANSPARENT,ALL_SIDES);
        //La flecha se coloca en el nivel facil
        if(message == "1")
            llSetLocalRot(facil);
        //La flecha se coloca en el nivel medio
        if(message == "2")
            llSetLocalRot(medio);
        //La flecha se coloca en el nivel dificil
        if(message == "3")
            llSetLocalRot(dificil);
        if (message == "reset")
            llSetLinkTexture(LINK_SET,TEXTURE_TRANSPARENT,ALL_SIDES);
    }
}
```

Figura 7: script de la flecha

El *script* de la caja con los temas funciona de manera similar, cuando el alumno toca uno de los cubos este se eleva y manda un mensaje al *script* del panel indicando que ha sido elegido. Así cuando el usuario pulsa el botón de comenzar se envía un mensaje con toda la información necesaria al *script* del crucigrama. El formato del mensaje mandado es el siguiente: comenzar&id_avatar&tema&nivel, se necesita

enviar el identificador del avatar para posteriormente almacenar los resultados obtenidos en el alumno correcto.

Para ilustrar el funcionamiento de la aplicación desde que el alumno ha elegido tema y dificultad hasta que se muestran todas las casillas que forman el crucigrama en la matriz se ha incluido la figura 8.

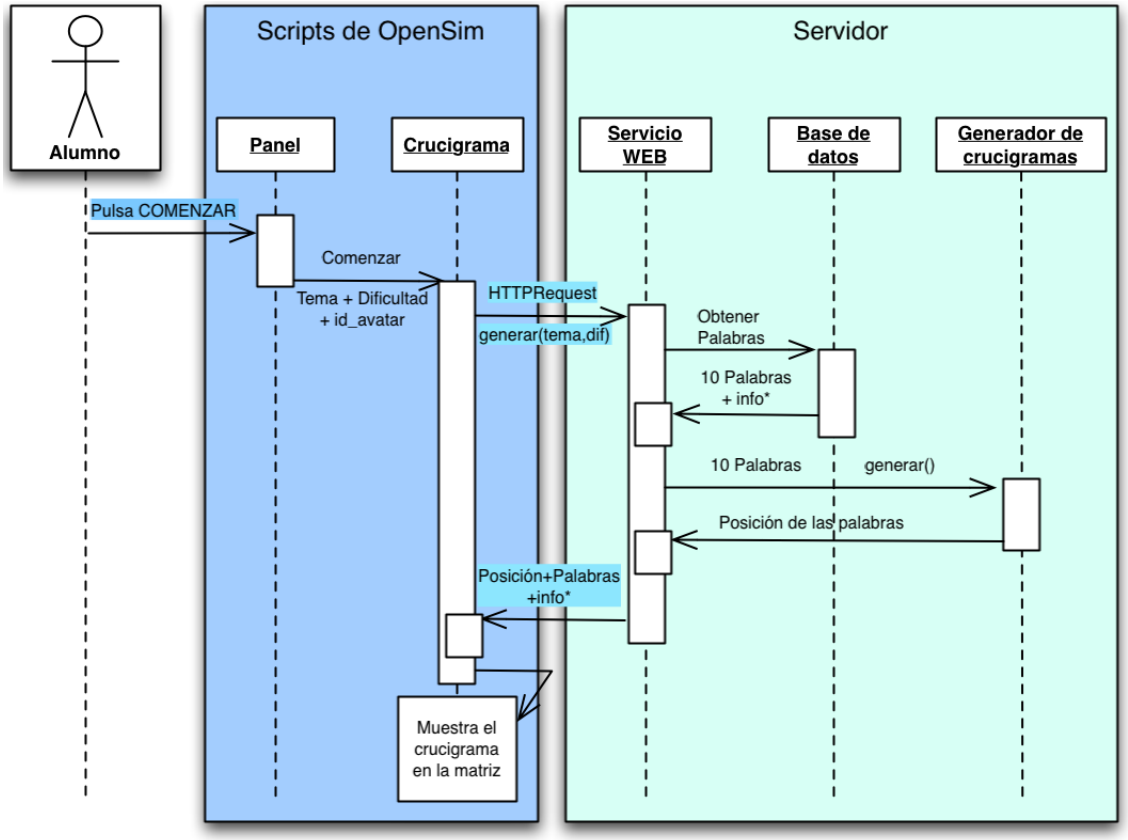


Figura 8: Diagrama de secuencia para generar un crucigrama

Una vez recopilada la información de dificultad y tema, el *script* del crucigrama hará una petición al servicio web tal y como se muestra en la figura 9. El método *llHTTPRequest* manda una trama HTTP con las características especificadas como parámetros, en este caso se envía a través del método *POST* y el tipo de contenido es formulario codificado[6]. En el cuerpo de la trama irá la información

[6] *application/x-www-form-urlencoded*

necesaria para la generación del crucigrama con el formato de un formulario http: método=generar&tema=2&dificultad=1.

```
//Mandamos la petición al servidor para que genere el crucigrama
 myRequest_generar = llHTTPRequest("http://[redacted].[redacted].7.234/Proyecto/crucigrama/init.php",
                                   [HTTP_METHOD, "POST",HTTP_MIMETYPE, "application/x-www-form-urlencod
                                   "metodo=generar&tema="+tema+"&dificultad="+dificultad);
```

Figura 9: Petición HTTP en LSL

El programa *init.php* recibirá la petición desde OpenSim obteniendo los parámetros de dificultad y tema. Seguidamente se solicitan a la base de datos diez palabras al azar que cumplan los requisitos de tema y nivel de dificultad. Para la dificultad lo que se hace es escoger un cierto número de palabras catalogadas con una dificultad. Por ejemplo, si se eligió un crucigrama fácil se escogen 5 palabras fáciles, 3 medias y 1 difícil, sin embargo si la dificultad elegida es difícil se escogen 2 fáciles, 3 medias y 5 difíciles. Para llevar a cabo todo este proceso es necesario configurar la conexión a la base de datos en PHP y hacer las consultas necesarias, en la figura 10 se puede ver un ejemplo de una de las consultas realizadas.

```
//Seleccionamos al azar las preguntas f□ciles
$resultado = mysql_query("SELECT Numero, Respuesta, Pregunta, Ayuda1, Ayuda2, Puntuacion1, Puntuacion2 FROM palabras
                          WHERE Dificultad = 1 and Tema=$tema ORDER BY RAND() LIMIT $num_faciles");
```

Figura 10: Consulta MySQL

Una vez obtenidas las palabras que se van a utilizar, init.php se las envía a *php crossword generator* para que este haga los cruces. Cuando llega a una solución devuelve la posición de la primera letra de cada palabra en coordenadas x-y[7] y si es horizontal o vertical. El siguiente paso es mandar a OpenSim la respuesta a la petición con toda la información necesaria en un formato determinado[8] para que sea posible dibujar el crucigrama y utilizar todas las funcionalidades.

La respuesta se captura en un evento llamado *http_response*, en la figura 11 se puede ver el procedimiento. Se filtra la respuesta recibida gracias a *request_id* que nos permite identificar a que petición pertenece y así poder realizar una acción u otra. En la figura 11 se observa que en el *script* también se realiza la petición de corregir el crucigrama al servicio web, pero esto lo explicaremos más adelante en uno de los apartados siguientes.

[7] posición inicial en la matriz 15x15.

[8] número&respuesta&X&Y&H/V&pregunta&ayuda1&ayuda2&penalización1&penalización2.

```
//Evento que se activa cuando se recibe una trama HTTP
http_response(key request_id, integer status, list metadata, string body){
    if (request_id == myRequest_generar){
        //Genera el crucigrama a partir de la informacion mandada por el servidor
        generar(body);
        llInstantMessage(avatar_key,"Terminado");
        llInstantMessage(avatar_key,body);
    }
    else if(request_id == myRequest_corregir){
        llInstantMessage(avatar_key,"Se ha enviado correctamente la informacion de tu crucigrama");
    }
}
```

Figura 11: Código en LSL para recibir las respuestas de las peticiones HTTP

Cuando el evento *http_response* captura la respuesta con toda la información necesaria para generar el crucigrama esta se pasa a una función llamada precisamente *generar*. Para la visualización del crucigrama se tiene una matriz de 15x15 cubos que están inicialmente transparentes, una vez recibidas las posiciones de todas las palabras lo que hace la función *generar* es cambiar las propiedades físicas de los cubos haciéndolos visibles. La información relacionada con cada pregunta se guarda en listas, es decir que si tenemos diez palabras diferentes en el crucigrama hay una lista de tamaño diez para almacenar las preguntas, en la posición uno de la lista está la pregunta para la primera palabra, en la segunda posición de la lista la pregunta de la segunda palabra y así sucesivamente. Lo mismo pasará con el resto de información: el identificador de la pregunta, la primera ayuda, la segunda ayuda, la penalización para la primera ayuda y la penalización para la segunda ayuda, de esta manera se tiene localizado todo lo necesario para cada pregunta. Una vez cambiadas las propiedades de todos los cubos que van a formar parte del crucigrama se muestran las preguntas en un cuadro de texto relacionadas a través de un número, así el avatar puede identificar qué pregunta pertenece a qué palabra. En la figura 12 se puede apreciar como queda finalmente un crucigrama recién generado.

2.4.2. Escribir sobre el crucigrama

Cuando el crucigrama aparezca el alumno tratará de resolverlo y para ello tiene que escribir sobre los cubos que representan las casillas del crucigrama. La idea general es que el avatar del alumno le "diga" directamente al crucigrama lo que quiere escribir a través del chat.

Como hemos visto en el apartado de generar crucigrama existe una manera de comunicarse con los objetos a través del evento *listen*. Si recordamos de lo

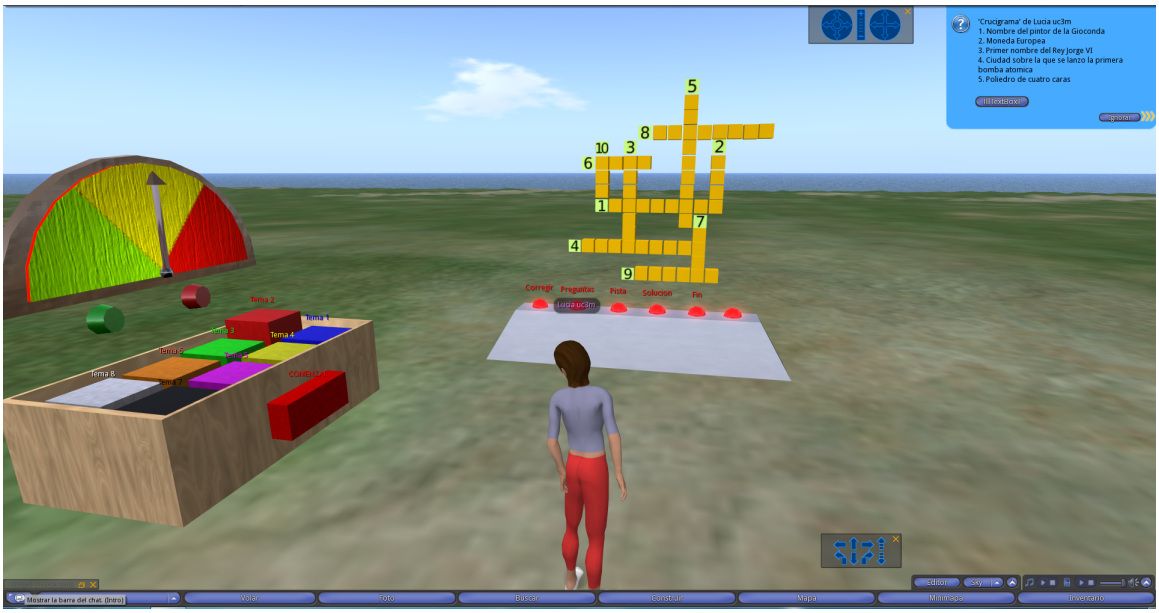


Figura 12: Crucigrama recién generado

anteriormente explicado el panel le enviaba una serie de información al *script* del crucigrama, concretamente la información enviada era el tema elegido, la dificultad elegida y el identificador del avatar, por lo tanto el *script* del crucigrama puede filtrar los mensajes que le llegan a través de este identificador y así solo reaccionar ante mensajes del avatar que generó el crucigrama.

El alumno tiene varias maneras de escribir sobre el crucigrama, la primera de ellas es pinchar sobre un número, entonces se seleccionará la palabra que pertenece a ese número resaltando los cubos en un color más brillante y desplazándose ligeramente hacia delante. Con este efecto visual al alumno le queda claro que palabra está resolviendo en cada momento. Una vez resaltada la palabra tan solo hay que escribir en el chat de OpenSim la respuesta y pulsar la tecla ENTER como si se enviara un mensaje normal, automáticamente el *script* detectará que ha ocurrido un evento y lo capturará a través de *listen*. Al tratarse del avatar que generó el crucigrama, este recogerá el mensaje y escribirá una letra en cada cubo de la palabra. Si la palabra escrita tiene una longitud mayor que cubos en el crucigrama simplemente se escribirá hasta donde se pueda. En la figura 13 se puede observar una palabra recién escrita siguiendo este proceso.

También existe la posibilidad de cambiar una letra en concreto, para ello tan solo hay que pinchar sobre el cubo y seguir el mismo proceso anterior.

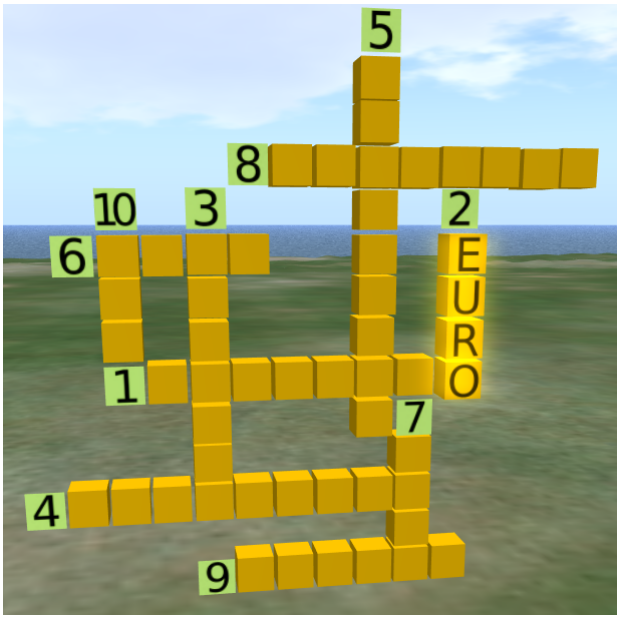


Figura 13: Palabra escrita en el crucigrama

Tan solo el avatar del alumno que generó el crucigrama será capaz de escribir sobre él, si otro avatar intenta realizar cualquier acción en un crucigrama ya generado este le ignorará.

Las letras sobre los cubos son texturas importadas a OpenSim, existe una función llamada *llSetLinkTexture*[9] que permite cambiar la textura del *prim* que le indiques dentro del conjunto que forma el objeto con el *script*. La textura correspondiente a la letra pulsada por el estudiante se encuentra por defecto en el inventario del propio avatar y se identifica, como el resto de elementos en OpenSim, a través de una llave o *key*.

2.4.3. Pedir pistas

En el crucigrama también están contempladas las pistas, así si un alumno no logra resolver alguna(s) pregunta(s) del crucigrama puede pulsar el botón de pista que hay sobre la mesa para que se revelen algunas de las letras que componen la palabra. Se pueden pedir dos pistas como máximo y cada una de ellas tiene una penalización sobre la puntuación final.

[9] *llSetLinkTexture(integer link_number, key texture_id, integer face)*

Como se explicó anteriormente, al generar el crucigrama se guardó toda la información recibida en listas ordenadas para poder acceder a ella cuando alguna funcionalidad concreta lo requiera. En este caso se necesita acceder a las listas que almacenan ambas ayudas, en total dos listas. Si por ejemplo el alumno pide la primera pista para la palabra ocho, el *script* accede a la posición ocho de la lista *ayuda1* y si pidiera una segunda ayuda para esta misma palabra el *script* accedería a la posición ocho de la lista *ayuda2*. Existe una lista llamada *pistas* de longitud diez con todas sus posiciones inicialmente a cero, así cuando se pide una pista se suma uno a la posición de la palabra que corresponda para posteriormente utilizar esta información a la hora de calcular la puntuación de cada palabra como se explica en la sección 2.4.5

Las ayudas tienen un formato predefinido para que el *script* del crucigrama pueda detectar que letras se quieren revelar. Las letras que no quieran mostrarse de la palabra deben sustituirse por el símbolo "_" y dejar tan solo las letras que se quieran mostrar en la pista. Todo esto se configura a través de la página web como veremos en la segunda parte.

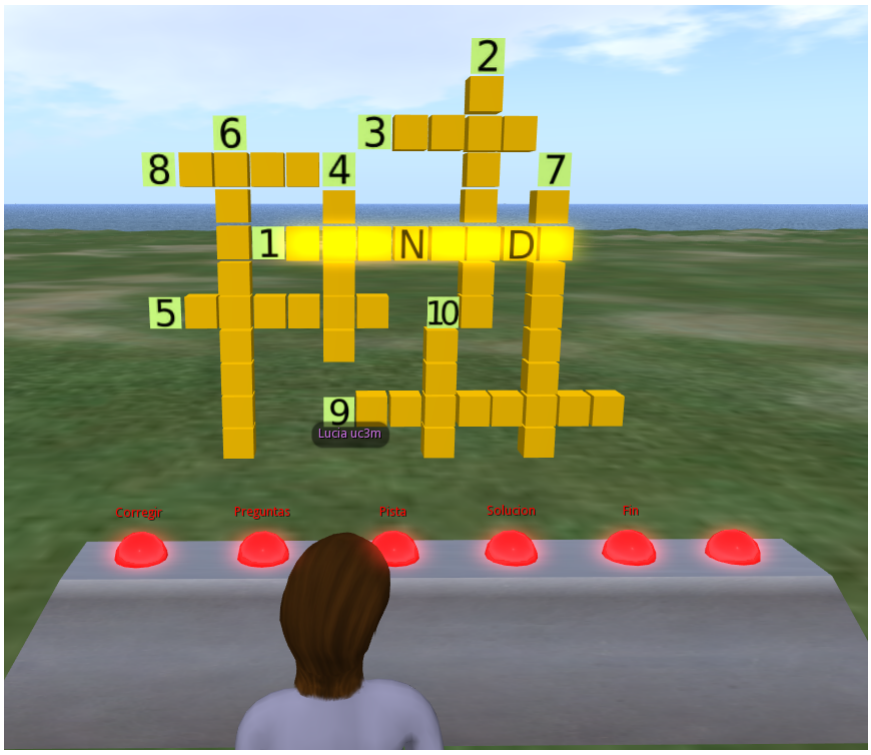


Figura 14: Pedir pistas en el crucigrama

En el ejemplo del figura 14 se ha pedido la pista para la palabra uno. La respuesta correcta es LEONARDO y al pedir pista se mostraron las letras N y D por lo tanto el formato para este caso en concreto es: _ _ _N_ _ _D_. Se puede pedir otra pista que mostrará otras letras de la solución o no mostrará ninguna dependiendo de lo que el profesor haya decidido al introducir la palabra en el sistema. Si se intentara pedir una tercera pista se mostraría un mensaje a través del chat que le indicaría al alumno que ya ha agotado las pistas disponibles, concretamente el objeto del crucigrama le diría al alumno: Ya has pedido dos pistas para esa palabra.

Los mensajes enviados por el crucigrama se mandan tan solo al avatar que está resolviendo el crucigrama como mensajes privados a través de la función *llInstantMessage(key id_avatar, string message)*, de esta forma no se molestaría a otros compañeros que pudieran estar haciendo sus respectivos crucigramas.

De aquí en adelante todas las funcionalidad explicadas comienzan pulsando un botón, como ya se explicó antes se puede capturar este evento a través de *touch_start* e identificar el botón que se pulsó gracias al parámetro *link_number*.

2.4.4. Ver preguntas

Aunque el panel con las preguntas aparece en la generación del crucigrama se puede dar el caso que un alumno cierre su panel y no pueda ver las preguntas correspondientes a su crucigrama, es por esta razón que se ha incluido un botón para volver a visualizar las preguntas de nuevo.

Como desde la generación del crucigrama tenemos toda la información guardada en listas, para este caso tan solo hay que consultar a la lista preguntas y crear un cuadro de texto con todas ellas. La función para generar un cuadro de texto es *llTextBox(key id_avatar, string message)*, igual que los mensajes privados que envía el crucigrama, el cuadro de texto solo podrá verlo el alumno que lo está resolviendo.

2.4.5. Corregir el crucigrama

Una vez finalizado el crucigrama el alumno pulsará el botón corregir para ver la puntuación que ha obtenido y que palabras ha puesto bien. Cuando esto ocurra el

script del crucigrama calculará la puntuación teniendo en cuenta las penalizaciones por pista de cada palabra, una respuesta perfecta puntuará 10 puntos, así un crucigrama perfecto serían 100 puntos. Una vez hecho esto se marcarán las palabras que están bien cambiando sus casillas al color verde y las que están mal o sin solución en color rojo. La puntuación conseguida se le dirá al alumno por el chat y se mandará al servicio web para que persista la información.

Para explicar todo el procedimiento que se lleva a cabo para la corrección del crucigrama se ha incluido la figura 15 que representa el díagrama de flujo desde que se captura el evento cuando el alumno pulsa el botón corregir hasta que se obtiene la nota total obtenida y se manda la información necesaria al servicio web. Todo este proceso se puede llevar a cabo en OpenSim porque se guardó la información de cada palabra en listas cuando fue recibida desde el servicio web al generar el crucigrama. Antes de comenzar a explicar el flujo de la aplicación es conveniente aclarar la notación utilizada:

- palabras[]: Lista donde se encuentra la solución de cada pregunta.
- penalizacion1[]: Lista donde se encuentran los puntos de penalización por utilizar la primera pista de cada pregunta.
- penalizacion2[]: Similar a penalizacion1[] solo que almacena los puntos de penalización por utilizar la segunda pista.
- respuesta_i: Se corresponde con la respuesta que ha introducido el alumno para la pregunta "i".
- pistas[]: Lista donde se encuentra el número de pistas que ha utilizado el alumno para cada palabra.
- notas[]: Lista donde se guarda la nota obtenida en cada una de las preguntas.
- nota_total: Es la nota obtenida en el crucigrama.

Cuando el alumno pulsa el botón corregir, el *script* del crucigrama captura el evento y reconoce que botón ha sido pulsado, a partir de aquí comienza el flujo mostrado en la figura 15. Se recorre la lista palabras comparando cada uno de las soluciones con la respuesta introducida por el alumno. Si la respuesta es correcta entonces se pasa a comprobar el número de pistas que han sido utilizadas para resolverla. Si no se ha utilizado ninguna pista se almacena un diez, la nota máxima, en la posición de la pregunta correspondiente, si se ha utilizado una pista se resta

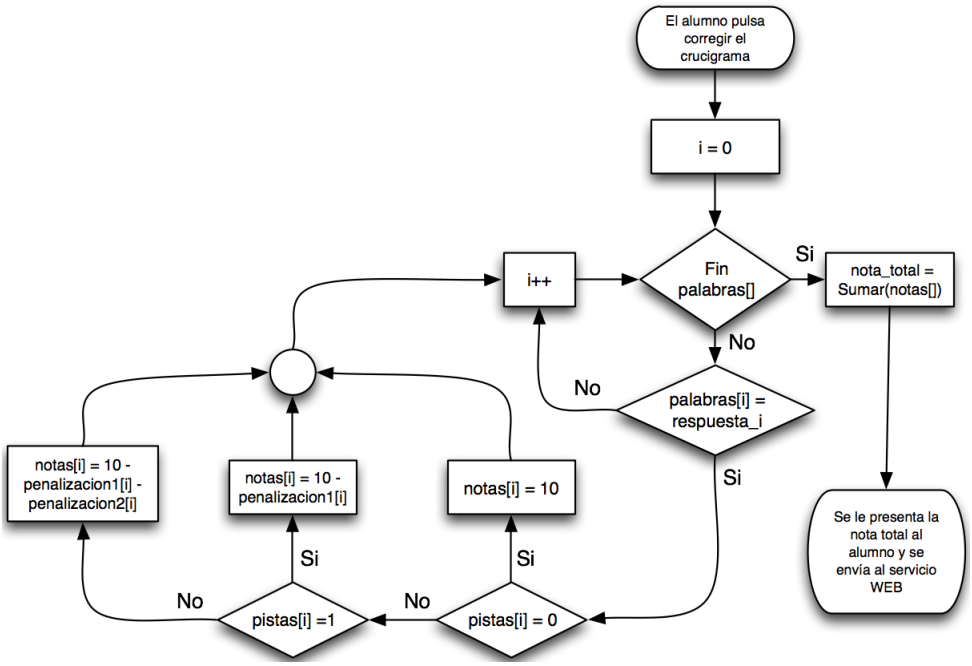


Figura 15: Diagrama de flujo de la funcionalidad corregir

la penalización para la primera pista a la puntuación máxima y se almacena, y si se han utilizado dos pistas se restan ambas penalizaciones a la nota máxima para posteriormente almacenarla como en los casos anteriores. Después de esto se sigue interando el mismo procedimiento hasta llegar al final de la lista preguntas, es decir, hasta que se hayan comprobado todas las respuestas del alumno. Por último se suman todas las posiciones de la lista notas para obtener la puntuación final del crucigrama.

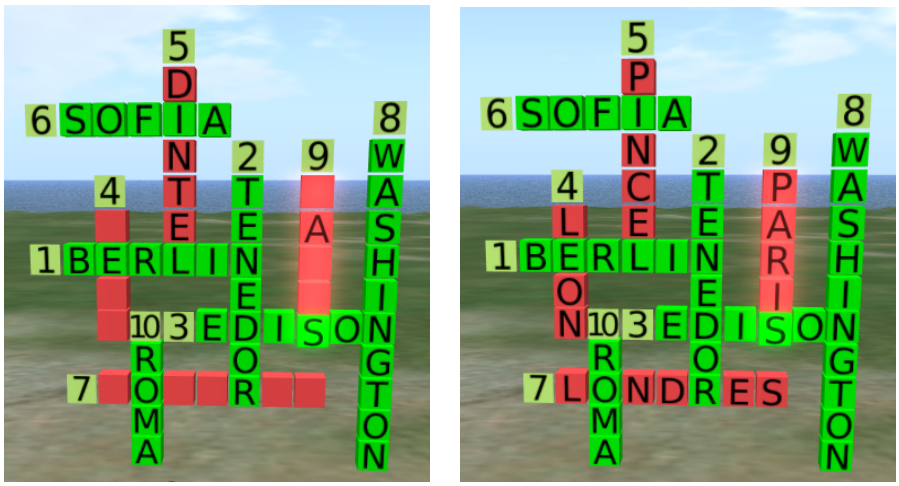
Una vez realizadas todas las operaciones necesarias en OpenSim se envía la información al servicio web para que la almacene en la base de datos además de cambiar el color de las casillas a rojo o verde dependiendo de si son correctas o no.

2.4.6. Solución del crucigrama

El crucigrama también incluye un botón para poder ver las respuestas correctas sobre el crucigrama. Para utilizar esta función el alumno tendrá que corregir previamente su crucigrama para así mandar al servidor la puntuación que ha conseguido, evitando que ningún alumno haga trampas pidiendo primero la solución

y corrigiendo después.

El funcionamiento es muy sencillo ya que se tienen almacenadas todas las palabras en una lista, tan solo hay que recorrerla mientras se dibujan las letras sobre los cubos. En la figura 16 se puede apreciar la solución del crucigrama.



(a) Crucigrama corregido (b) Solución del crucigrama

Figura 16: Crucigrama corregido y solución

2.4.7. Finalizar crucigrama

Existe un botón sobre la mesa con la etiqueta “Fin”. Al pulsar este botón lo que ocurre es que se resetea el *script* del crucigrama llevando al objeto a su estado inicial. Lo que el alumno apreciará es que desaparecen los cubos del crucigrama que estaba haciendo y tendrá que volver a generar otro si quiere continuar resolviendo crucigramas.

2.5. Diseño de la base de datos

Esta sección podría pertenecer a ambas partes del presente proyecto, tanto al crucigrama como a la página web, ya que la base de datos es utilizada por ambas. El crucigrama es el encargado de rellenar algunas de las tablas existentes con los datos recopilados cuando los alumnos resuelven sus crucigramas, y la página web consulta y sintetiza estos datos para mostrarlos de una manera útil. Como

la explicación de la estructura de la base de datos facilita la comprensión de lo expuesto anteriormente, se ha decidido aclarar el diseño en esta sección.

Existen cuatro tablas en total: usuarios, palabras, propuestas y evaluación. En la figura 17 se puede apreciar la estructura de la base de datos y las relaciones entre las tablas. Tanto la tabla propuestas como la tabla evaluación se relacionan con la tabla usuarios a través de una clave foránea. A continuación explicaremos más en detalle cada una de las tablas.

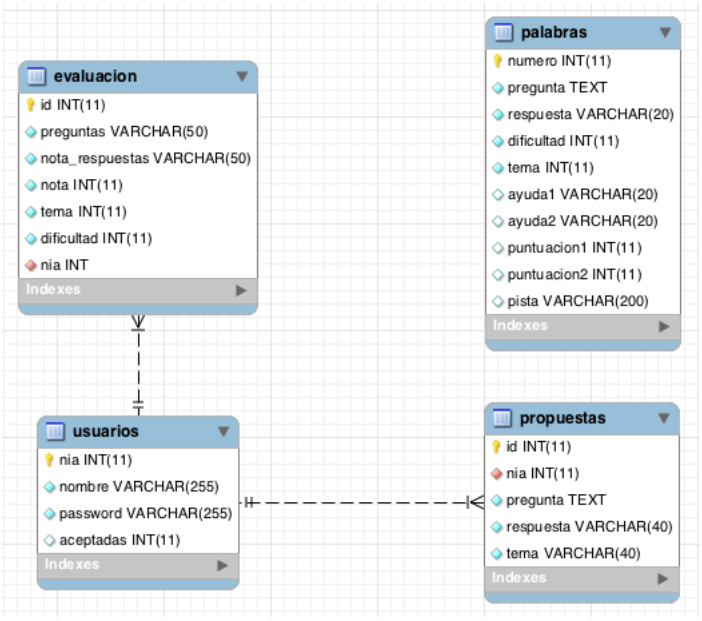


Figura 17: Estructura de la base de datos

2.5.1. La tabla usuarios

En esta tabla se almacenan todos los usuarios del sistema completo, es decir, alumnos y profesores. Estos son los diferentes campos que se almacenan para cada usuario:

- nia: El número de identificación de alumno de cada usuario. En el caso del profesor se le asignará un identificador diferente, hemos elegido el 1000. Es la clave primaria de la tabla.
- nombre: Nombre y apellidos del usuario. Tiene que ser único y no nulo.

- password: Contraseña del usuario para la página web. No puede ser nulo.
- aceptadas: Numero de propuestas que han sido aceptadas. Este elemento solo tiene sentido si el usuario se trata de un alumno, en el caso del profesor será nulo.

2.5.2. La tabla palabras

En la tabla palabras se almacenan todas las posibles preguntas que pueden aparecer en un crucigrama. Contiene los siguientes campos:

- número: Es la clave primaria y el identificador de la tabla. Se asigna automáticamente al incluir una nueva pregunta en la tabla.
- pregunta: Cadena de caracteres que representa el enunciado de la pregunta del crucigrama. No puede ser nulo.
- respuesta: Cadena de caracteres que representa la palabra que aparecerá en el crucigrama. No puede ser nulo.
- dificultad: Entero que identifica el nivel de dificultad de la pregunta. Se han definido tres niveles de dificultad, 1:Fácil, 2:Medio y 3:Dificil. No puede ser nulo.
- tema: El tema al que pertenece la pregunta. No puede ser nulo.
- ayuda1: Es la primera pista que se le dará al alumno. Viene definida como se explicó en la sección 2.4.3.
- ayuda2: Es la segunda pista que se le dará al alumno. Está definida igual que ayuda1.
- puntuacion1: La penalización que supondrá pedir la primera pista. Una palabra perfecta puntúa 10, por lo tanto esta cifra debe de estar entre 0 y 10.
- puntuacion2: La penalización que supondrá pedir la segunda pista. La suma entre penalizacion1 y penalizacion2 no debe superar los 10 puntos.
- pista: Se puede guardar una pista para cada pregunta que será mostrada en la página web.

2.5.3. La tabla evaluación

En esta tabla se almacenan los resultados obtenidos por los alumnos al hacer los crucigramas. Cada entrada se corresponde con un crucigrama:

- id: Es la clave primaria de la tabla. Se asigna automáticamente un id a cada entrada ya que es auto incremental.
- preguntas: Es una cadena de caracteres en un formato definido que contiene los números de las preguntas que han compuesto el crucigrama.
- nota_respuestas: También es una cadena de caracteres construida con el mismo formato que preguntas. Almacena la nota conseguida en cada una de las preguntas almacenadas en el campo anterior.
- nota: Es la nota obtenida en el crucigrama. Se puede calcular también sumando todas las puntuaciones obtenidas en cada una de las preguntas.
- tema: El tema al que pertenece el crucigrama.
- dificultad: La dificultad del crucigrama.
- nia: El número de identificación del alumno que resolvió el crucigrama. Es una clave foránea que apunta a la clave primaria de la tabla usuarios.

El formato para guardar las preguntas que salieron en el crucigrama y las puntuaciones que obtuvo el alumno en cada una de ellas es simple, se separan con comas los diez números de las preguntas que forman el crucigrama y se guarda la cadena completa. Por ejemplo, en el campo preguntas podría almacenarse la siguiente cadena: *1,2,3,4,5,6,7,8,9,10*, así si se necesitan mostrar las características de las preguntas que aparecieron en un crucigrama en concreto, tan solo hay que recorrer esa cadena teniendo en cuenta como separador la coma y consultar la tabla palabras. Por otro lado se guardan las puntuaciones de las preguntas con el mismo formato en el campo nota_respuestas: *10,10,4,2,8,7,6,10,0,8*. Como un crucigrama siempre está formado por diez palabras existe una relación inequívoca entre la nota y el número de pregunta. Toda esta información será sintetizada y presentada de una manera útil tanto a alumnos como profesores a través del portal web.

2.5.4. La tabla propuestas

En esta tabla se almacenan las propuestas hechas por los alumnos. El profesor podrá valorarlas e incluirlas entre las posibles preguntas que pueden aparecer en el crucigrama. Más adelante veremos como se hace.

- id: Clave primaria de la tabla. Se asigna automáticamente un id a cada entrada porque es auto incremental.
- nia: El número de identificación del alumno que mandó la propuesta. Es una clave foránea que apunta a la clave primaria de la tabla usuarios.
- pregunta: La pregunta propuesta por el alumno. No puede ser nulo.
- respuesta: La respuesta a la pregunta propuesta. Tampoco puede ser nulo.
- tema: El tema al que corresponde la pregunta propuesta. No puede ser nulo.

3. OpenWeb

A partir de ahora denominaremos a la página web, OpenWeb. En esta sección se van a exponer las funcionalidades y las tecnologías utilizadas para el desarrollo de la página web. El cometido de OpenWeb es sintetizar la información recogida en la resolución de los crucigramas y mostrarla de una manera útil tanto a alumnos como profesores.

3.1. Descripción de las tareas

A parte de resolver los crucigramas es interesante para los alumnos poder consultar su evolución y tener una plataforma de apoyo. También es interesante para el profesor hacer un seguimiento de sus alumnos además de obtener información acerca de las preguntas que están saliendo actualmente en los crucigramas. Todas estas posibilidades de interacción serán presentadas por OpenWeb a los usuarios del sistema. Se pueden resumir las tareas para alcanzar la funcionalidad deseada en los siguientes puntos:

- Iniciar sesión: Se debe desarrollar una primera vista para el inicio de sesión por parte de alumnos o profesores. Dependiendo del usuario se le mostrará posteriormente una vista u otra.
- Vista del profesor: Incluye todas las posibles funcionalidades que puede utilizar el profesor, además en esta vista se muestran las preguntas propuestas por los alumnos que tiene pendientes de revisión. A continuación se enumeran las diferentes opciones que se desea que el profesor tenga accesibles.
    1. Consultar las estadísticas de un alumno en concreto. Podrá filtrar por tema y dificultad del crucigrama.
    2. Consultar las estadísticas de las preguntas. Podrá filtrar por tema y dificultad del crucigrama.
    3. Ver el ranking de los alumnos con las mejores notas. También se puede filtrar por temas y dificultades.
    4. Ver las propuestas que han realizado los alumnos con la posibilidad de añadirlas a las posibles preguntas o eliminarlas.
    5. Añadir preguntas nuevas.
- Vista del alumno: Será parecida a la del profesor pero con diferentes funcionalidades de más interés para el alumno, además se le mostrarán las propuestas

que han sido aceptadas por el profesor. A continuación están enumeradas las diferentes opciones que tendrá disponibles el alumno:

1. Consultar sus estadísticas con la posibilidad de filtrar por tema y dificultad.
2. Consultar las preguntas que han salido en sus crucigramas eligiendo un tema y una dificultad.
3. Ver las pistas para cada una de las preguntas.
4. Ver el ranking por temas y dificultades.
5. Proponer preguntas al profesor.

3.2. Herramientas y tecnologías utilizadas

Se decidió utilizar Java/J2EE para la implementación de la página de servicio web. A continuación se describen una a una todas las tecnologías y herramientas utilizadas para el desarrollo de OpenWeb.

3.2.1. Maven

Maven[11] es una herramienta para la construcción y gestión de proyectos en Java. La configuración se realiza a través de un fichero xml llamado *pom.xml* donde se define todo lo que necesita el proyecto para funcionar. Maven maneja las dependencias del proyecto, compila, empaqueta y ejecuta tests. También permite la utilización de plugins que realizan otras acciones sobre el proyecto.

La utilidad más destacada de Maven en este proyecto es el manejo de dependencias. Existen multitud de repositorios en la red de los cuales se obtienen los paquetes necesarios para el funcionamiento de un proyecto. Hay una carpeta local llamada *.m2* donde se descargan las dependencias utilizadas, en caso de que no se encuentren ya descargadas, y se almacenan los proyectos creados. Si el elemento que se quiere utilizar no está en ningún repositorio remoto se puede descargar manualmente y guardar en esta carpeta. El formato para la definición de una dependencia viene descrito en la figura 18. Todos los proyectos realizados con Maven tienen un *groupId* que sería como el nombre de un paquete que contiene un conjunto de elementos, el *artifactId* indica el nombre en concreto del elemento dentro del paquete y por último *version* indica la versión del elemento que se desea utilizar.

Es obligatorio definir estas tres etiquetas cuando se crea un proyecto con Maven para poderlo localizar y facilitar su utilización en futuros proyectos[12].

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>2.5.6</version>
    <scope>compile</scope>
</dependency>
```

Figura 18: Definición de una dependencia en Maven

A parte de la utilidad de las dependencias se ha utilizado un plugin para desplegar el proyecto directamente en un servidor remoto. Como se aprecia en la figura 19 hay que configurar la dirección IP del servidor, además hay que guardar un perfil válido de usuario en el fichero *settings* de Maven[10] que en este caso se identifica dentro de la etiqueta *server* como tomcat.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>tomcat-maven-plugin</artifactId>
            <configuration>
                <server>tomcat</server>
                <url>http://[redacted].7.234:8080/manager</url>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Figura 19: Utilización de plugins en Maven

3.2.2. Spring MVC Framework

Antes de entrar a definir lo que es el framework Spring MVC y para qué sirve vamos a explicar algunos de los conceptos utilizados en el desarrollo de este proyecto, como el patrón de diseño MVC o la inyección de dependencias, ambas utilizadas en Spring MVC.

[10]Este fichero se encuentra en el mismo lugar que la carpeta .m2 donde se descargan las dependencias

Para el desarrollo de la aplicación se ha utilizado el patrón Modelo-Vista-Controlador (MVC) que distingue entre tres tipos de elementos:

- *Modelo:* es la parte de la aplicación que se encarga de gestionar los recursos persistentes; en nuestro caso, se trata de la interfaz de la base de datos para el resto del sistema. Es la única parte de la aplicación que debería tener un acceso directo a estos recursos para garantizar la modularidad.
- *Vista:* es la interfaz que se presenta al usuario para interactuar con el contenido de la aplicación. También es responsable de mostrar los datos que se almacenan en el modelo, aunque no está permitido hacerlo de forma directa.
- *Controlador:* es la lógica propiamente dicha de la aplicación. Contiene el funcionamiento real que debe tener lugar cuando se realiza una acción en la vista, y obtiene los datos del modelo para que la vista pueda utilizarlos. Por tanto, el controlador es el nexo de unión entre vista y modelo.

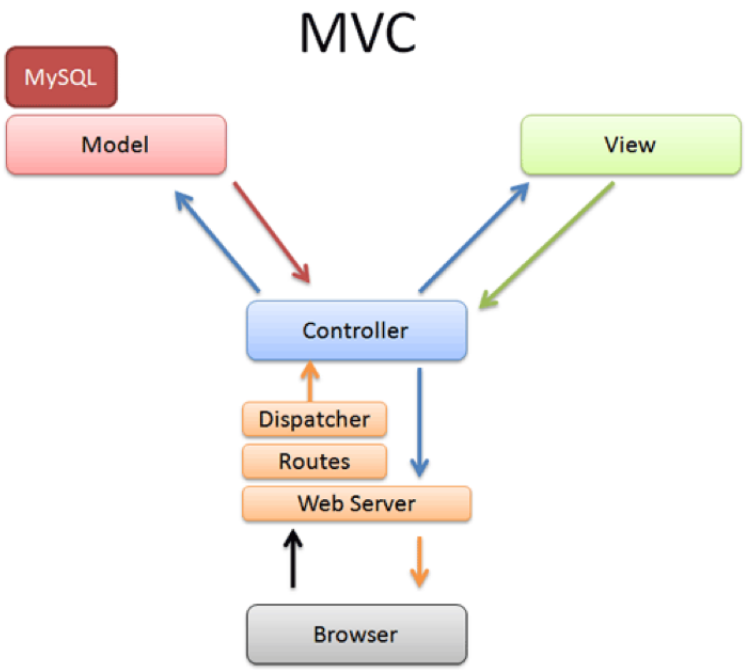


Figura 20: Esquema de un aplicación desarrollada con el paradigma MVC

Como puede observarse en la figura 20, el usuario, a través de su navegador web, al interactuar con la vista que tiene presente, produce el envío de una petición al servidor web que tiene alojada dicha aplicación[13]. Éste la hace llegar al

controlador pertinente, que se encarga de gestionar el paso de información entre un modelo determinado y la vista a mostrar. De esta manera las aplicaciones adquieren una modularidad que simplifica la comprensión del desarrollo y permite la reutilización de alguna de las partes.

También se ha utilizado el patrón de inyección de dependencias[14] donde los objetos son suministrados a las clases en vez de que sea la propia clase la que crea los objetos. Una consecuencia directa de este patrón es que las instancias inyectadas no se generan desde el código sino que se generan, en este caso, a través del framework. Por lo tanto se trabaja siempre a través de una interfaz, el framework utilizado es el encargado de gestionar el objeto y crearlo. En este sentido se pierde el control absoluto de la aplicación y se trabaja con una caja negra donde se saben las entradas y las salidas pero no el funcionamiento exacto de su interior.

Spring es una plataforma que nos proporciona una infraestructura que actúa de soporte para desarrollar aplicaciones Java. Está dividido en varios en módulos diferentes, en este caso se ha utilizado el módulo de Spring MVC dadas las características de este proyecto. El núcleo de Spring realiza la inyección de dependencias, y el módulo Spring MVC implementa una arquitectura Modelo-Vista-Controlador. Una de las ventajas de utilizar este framework es que desaparece de nuestro código toda la creación de objetos y paso de dependencias, con lo que obtenemos un código más limpio, pero por otro lado necesitaremos de algunos ficheros xml de configuración. Además Spring motiva la programación a través de interfaces con lo que se logra un máximo desacoplamiento entre clases.

Es interesante definir ahora cual es el flujo básico que lleva a cabo el framework[15]. Para apoyar la explicación se ha incluido la figura 21:

1. La petición llega a DispatcherServlet.
2. Ahora DispatcherServlet tiene que encontrar el controlador que va a manejar la petición a través de la url. Para ello se utiliza el HandlerMapping.
3. Una vez encontrado el controlador, DispatcherServlet le dejará tratar la petición. El controlador llevará a cabo las acciones a través de las capas inferiores de la aplicación (servicios, acceso a bases de datos, etc...).
4. Cuando el controlador termine las acciones pertinentes, le devolverá a DispatcherServlet un objeto del tipo ModelAndView que contiene, por un lado la información obtenida de las capas inferiores de la aplicación, también llamado modelo, y por otro la vista que se quiere mostrar a continuación donde irá contenida esta información.

5. A continuación el DispatcherServlet debe asociar el nombre de la vista devuelto por el controlador a una vista concreta a través del ViewResolver.
6. Por último, el DispatcherServlet pasa la información, es decir, el modelo, a la vista en concreto para su presentación.

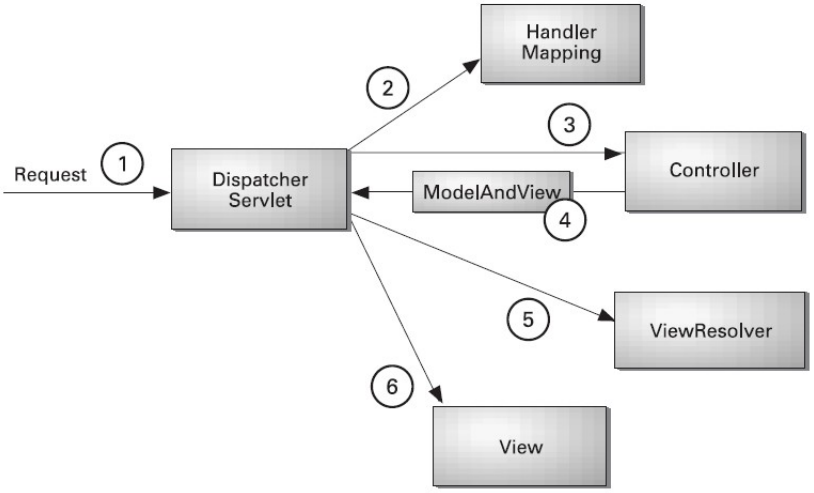


Figura 21: Funcionamiento de Spring MVC

Es necesaria la creación de ciertos ficheros xml para la correcta configuración de la aplicación. A continuación se presentará de una manera general cuales son las consideraciones necesarias para desarrollar un proyecto utilizando este framework.

Para empezar hay que incluir el servlet de Spring en el fichero de configuración *web.xml*, este servlet se encargará de gestionar las llamadas a las páginas web internamente. De esta manera todas las url que terminen por .htm serán servidas por este servlet tal y como se muestra en la figura 22, abstrayéndonos de su funcionamiento en concreto[16]. Solo hay que preocuparse de definir los controladores como veremos más adelante.

```
<servlet>
  <servlet-name>openweb</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>openweb</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

Figura 22: Configuración del servlet de Spring

A continuación se debe generar otro fichero de configuración que por convenio debe llamarse *nombre_del_servlet_de_Spring-servlet.xml* es decir que en nuestro caso como hemos llamado al servlet *openweb* el fichero debe de llamarse *openweb-servlet.xml*, este fichero se guarda en la misma carpeta que *web.xml*, WEB-INF. Este archivo de configuración le indicará al servlet que clases son los controladores de nuestra aplicación.

```
<bean name="/index.htm" class="uc3m.paginaweb.web.IndexController">
        <property name="webManager" ref="webManager" />
</bean>

<bean name="/alumno.htm" class="uc3m.paginaweb.web.AlumnosController">
        <property name="webManager" ref="webManager" />
</bean>

<bean name="/profesor.htm" class="uc3m.paginaweb.web.ProfesorController">
        <property name="webManager" ref="webManager" />
</bean>

<bean name="/formulario.htm" class="uc3m.paginaweb.web.FormularioController">
        <property name="webManager" ref="webManager" />
</bean>

<bean name="/accion.htm" class="uc3m.paginaweb.web.AccionesController">
        <property name="webManager" ref="webManager" />
</bean>
```

Figura 23: Configuración de los controladores en Spring

En la figura 23 se pueden ver los controladores que hay definidos para OpenWeb. Se indica a través de la etiqueta *name* la url que va a gestionar el controlador, en la etiqueta *class* se le indica la clase que va a desempeñar el papel de controlador y en *property* se ha definido el objeto que necesita ser inyectado al controlador, en este caso lo hemos llamado *webManager*. Así cualquier petición dirigida a *alumno.htm* será capturada por el servlet de Spring que lo mandará a su vez al controlador definido, en este caso *AlumnosController*.

Por último se necesita el fichero de configuración para la inyección de dependencias, por convenio se llama *applicationContext.xml*. En este archivo se le indica donde están las clases para crear dichos objetos. En la figura 24 se aprecia como se indica la clase para generar objetos de la propiedad *webManager* y este a su vez necesita de la inyección de otros objetos que vienen definidos en este mismo fichero.

```
<bean id="webManager" class="uc3m.paginaweb.service.WebManager">
        <property name="usuariosDao" ref="usuariosDao"/>
        <property name="crucigramasDao" ref="crucigramasDao"/>
        <property name="preguntasDao" ref="preguntasDao"/>
        <property name="propuestasDao" ref="propuestasDao"/>
        <property name="dataManager" ref="dataManager"/>
</bean>

<bean id="usuariosDao" class="uc3m.paginaweb.repository.UsuariosDao" />

<bean id="crucigramasDao" class="uc3m.paginaweb.repository.CrucigramasDao" />

<bean id="preguntasDao" class= "uc3m.paginaweb.repository.PreguntasDao"/>

<bean id="propuestasDao" class= "uc3m.paginaweb.repository.PropuestasDao"/>

<bean id="dataManager" class="uc3m.paginaweb.service.DataManager"/>
```

Figura 24: Configuración de los objetos inyectados

3.2.3. Java Persistence API

En el actual proyecto se ha utilizado la base de datos MySQL. Java trabaja con objetos en memoria que para poder almacenarlos en disco deben ser correlacionados (*marshalling*). JPA permite realizar dicha correlación de manera sencilla, abstrayéndonos de toda la conversión entre objetos y tablas. A esta conversión se le llama ORM (Object Relational Maping) y se configura a través de metadatos (mediante xml o anotaciones). En resumen, JPA nos permite persistir objetos y crear objetos a partir de tablas, a estos objetos se les llama entidades.

JPA[17] establece una interfaz común que es implementada por un proveedor de persistencia a nuestra elección, en este caso es Hibernate, siendo este el que realiza el trabajo pero siempre a través del API de JPA.

En la figura 25 se observa la clase usuario de la cual se instancian los objetos que actúan como entidades. La anotación *@Entity* informa al proveedor de persistencia que cada instancia de esta clase es una entidad, y en la anotación *@Table* se le pasa el nombre de la tabla en concreto a la cual pertenece dicha entidad. Los objetos entidad son POJOs (Plain Old Java Object), esto es objetos sencillos, en este caso tan solo poseen atributos y métodos get y set. Cada uno de los atributos se corresponde con el nombre de las columnas de la tabla, de esta manera el proveedor de persistencia es capaz de establecer la correspondencia entre una entrada de una tabla a un objeto o viceversa. Todas las entidades tienen que tener una identidad que las diferencie del resto, por lo que deben tener una propiedad marcada con la anotación *@Id*.

```java
package uc3m.paginaweb.domain;

import javax.persistence.*;


@Entity
@Table(name="usuarios")
public class Usuario {

	@Id
	private int nia;

	private String nombre;
	private String password;
	private int aceptadas;

	public int getNia() {
		return nia;
	}
	public void setNia(int nia) {
		this.nia = nia;
	}
	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
	public int getAceptadas() {
		return aceptadas;
	}
	public void setAceptadas(int aceptadas) {
		this.aceptadas = aceptadas;
	}
}
```

Figura 25: Entidad usuario

El concepto de persistencia implica el hecho de almacenar nuestras entidades en un sistema de almacenamiento como una base de datos, pero más allá del proceso de almacenamiento todo sistema de persistencia debe permitir recuperar, actualizar y eliminar dichas entidades. Estas cuatro operaciones se conocen como operaciones CRUD (Create, Read, Update, Delete) y JPA es capaz de manejarlas a través de la interfaz *EntityManager*.

Para que JPA pueda realizar todas estas operaciones sobre las entidades necesita un archivo de configuración XML llamado *persistence.xml*, el cual debe estar ubicado en el directorio META-INF de la aplicación. El archivo *persistence.xml* contiene información necesaria, como el nombre de la unidad de persistencia, las clases que deseamos que sean manejadas por el proveedor de persistencia, los pa-

```
<persistence-unit name="crucigrama">

<provider>org.hibernate.ejb.HibernatePersistence</provider>
        <class>uc3m.paginaweb.domain.Usuario</class>
        <class>uc3m.paginaweb.domain.Crucigrama</class>
        <class>uc3m.paginaweb.domain.Pregunta</class>
        <class>uc3m.paginaweb.domain.Propuesta</class>
```

Figura 26: Fragmento del fichero de configuración de persistencia

rámetros para conectar con nuestra base de datos o el proveedor de persistencia que vamos a utilizar. En la figura 26 se puede ver un fragmento del fichero *persistence.xml* utilizado en este proyecto donde se muestra el nombre de la unidad de persistencia (en este caso, *crucigrama*), el tipo de proveedor utilizado (Hibernate) y las clases que va a manejar el proveedor, que en este caso son cuatro: Usuario, Crucigrama, Pregunta y Propuesta.

A continuación se muestra un ejemplo de como persistir un objeto a través de la interfaz *EntityManager*. El método mostrado en la figura 27 pertenece a la clase que maneja los accesos a base de datos de la entidad usuario, y concretamente es el método encargado de buscar un usuario. La etiqueta *@Transacational* indica que el método es transaccional, pero esta etiqueta pertenece al framework de Spring MVC. El método necesita un parámetro que es el nia del alumno que se quiere buscar y devuelve un objeto Usuario. Lo primero que se hace es obtener la interfaz *EntityManager*, para ello se hace referencia a la unidad de persistencia creada en el fichero de configuración cuyo nombre es *crucigrama*. A continuación se indica que va a comenzar la transacción y se llama al método *find*, pasándole como parámetros la clase a la cual pertenece la entidad que se está buscando y el identificador de la entidad, que en este caso es el nia. Por último se finaliza la transacción y se cierra la interfaz *EntityManager*. En resúmen, lo que conseguimos con este método es obtener un objeto de la clase Usuario que podremos manejar en nuestra aplicación Java sin involucrarnos en el proceso de mapeo desde la base de datos.

3.2.4. JavaServer Pages

JavaServer Pages o JSP[18] es una tecnología Java que permite generar contenido dinámico en una página web. Permite utilizar fragmentos de código Java dentro de un documento HTML, para ello hay que cambiar la extensión de estos documentos a *.jsp* en vez de *.html*. Lo que ocurre cuando se abre una página JSP

```java
@Transactional
public synchronized Usuario buscarUsuario(int nia){
        entityManagerFactory = Persistence.createEntityManagerFactory("crucigrama");
        EntityManager entityManager = entityManagerFactory.createEntityManager();
        entityManager.getTransaction().begin();

        Usuario u = entityManager.find(Usuario.class, nia);

        entityManager.getTransaction().commit();
        entityManager.close();
        entityManagerFactory.close();

        return u;
}
```

Figura 27: Método para buscar un usuario a partir del nia

es que esta se convierte a un fichero Java, se compila y finalmente se carga, por ello la primera vez que una página JSP sea abierta tardará más en cargar que una HTML, pero la compilación solo ocurre una vez, así que después de la primera carga, la página no tardará en abrirse.

Para poder incluir código Java entre el código HTML que típicamente define una página web, hay que empotrarlo entre los símbolos "<%" y "%>" de la siguiente manera: *<% código Java %>*. En la figura 28 se aprecia un fragmento que muestra como se genera dinámicamente un menú desplegable dependiendo de los alumnos que haya registrados en la aplicación. La tecnología JSP nos permite que las opciones del menú sean dinámicas. Como se explicó en la sección 3.2.2 acerca de Spring MVC, el controlador pasa un modelo con los datos a la vista para que esta los represente, en el fragmento mostrado también se puede apreciar este detalle.

```jsp
<form:form action="profesor.htm" method="post">
    <div id="alumnos">
        <select name="alumnos" class="dk">
            <option selected> NIA -- NOMBRE DEL ALUMNO
                <%
                    Map<String, Object> model = (Map<String, Object>) request.getAttribute("model");
                    Map<Integer, String> alumnos = (Map<Integer, String>) model.get("alumnos");
                    Set<Integer> keys = alumnos.keySet();
                    for (Iterator<Integer> it = keys.iterator(); it.hasNext();) {
                        Integer nia = it.next();
                        String nombre = alumnos.get(nia);
                        out.print("<option value='" + nia + "'>" + nia + " -- " + nombre);
                    }
                %>
        </select>
    </div>
```

Figura 28: Fragmento de una página JSP

Por último solo queda decir que también se han utilizado páginas de estilo CSS para dar formato visual a las páginas web. Se puede resumir que para la "vista" del patrón MVC se ha utilizado CSS, HTML y JSP.

3.2.5. NetBeans 7.1.2

Es un entorno de desarrollo integrado (o IDE: Integrated Development Environment) libre hecho principalmente para el lenguaje Java. Existe un gran número de módulos para extenderlo integrando la gran mayoría de los frameworks de desarrollo actuales.

NetBeans tiene un conjunto de herramientas de mucha utilidad para desarrollar aplicaciones en Java, cabe destacar la capacidad de generar código para ciertos patrones de aplicación. Por ejemplo, si se tiene un proyecto que precisa de JPA para acceder a las bases de datos, NetBeans puede generar todas las entidades y los métodos CRUD. Existen muchas más posibilidades, como generar APIs REST, aplicaciones web, etcétera.

Por otro lado también es destacable el navegador de proyectos donde pueden verse los diferentes módulos de una aplicación. Dependiendo del tipo de aplicación que estemos desarrollando NetBeans mostrará una estructura u otra.

3.3. Estructura de la aplicación

La estructura de OpenWeb se muestra en la figura 29. Se han desplegado las carpetas para que sean visibles los ficheros más importantes que han sido mencionados en a sección anterior. Dentro de la carpeta Web-Pages se encuentran algunos de los ficheros xml de configuración, en concreto están dentro de la carpeta WEB-INF. Estos ficheros son los necesarios para configurar el framework Spring MVC. También se encuentra todo lo necesario para construir la vista: estilos, imágenes, scripts y las propias páginas jsp. En la carpeta Source Packages es donde se encuentran las clases java que constituyen el núcleo de la aplicación. Cabe destacar la organización por paquetes de las clases, quedando bien diferenciada la funcionalidad de cada paquete:

- domain: Donde se encuentran las entidades.
- repository: Están las clases que utiliza la interfaz *EntityManager* para el manejo de entidades.
- resources: Clases que contienen recursos necesarios para el funcionamiento de otras, como por ejemplo constantes.

- service: Están las clases que conectan los controladores con el modelo y realizan toda la lógica de negocio, es la capa con más peso de la aplicación.
- web: Donde se encuentran los controladores.

En Other Sources está el fichero de configuración de persistencia para JPA y en Project Files nos encontramos con el fichero *pom.xml* necesario para la configuración de Maven. La estructura de una aplicación es importante para la modularidad y comprensión de la misma, en el presente proyecto se ha intentado seguir una línea de diseño que fomentase la buena comprensión y la capacidad de división de la aplicación.

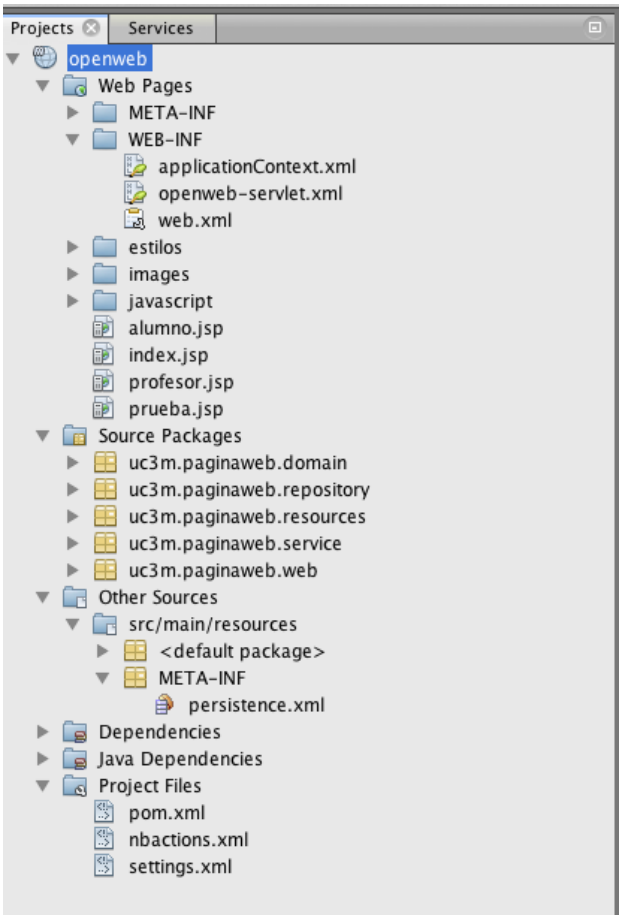


Figura 29: Estructura de la aplicación en NetBeans

En la figura 30 se aprecia más en detalle la estructura de clases que existe. La primera capa es la de las vistas solo accesible desde la capa de controladores,

después hay una capa intermedia entre el modelo y los controladores que es la capa de servicios, estas clases son las encargadas de hacer toda la lógica de negocio entregando al controlador la información del modelo ya procesada. Por último se tiene la capa de modelo, subdividida a su vez en otras dos capas, las clases encargadas del acceso a los datos y sus correspondientes entidades. Se ha respetado en todo momento el paradigma de desarrollo MVC, las clases del modelo nunca acceden directamente a la vista ni viceversa.

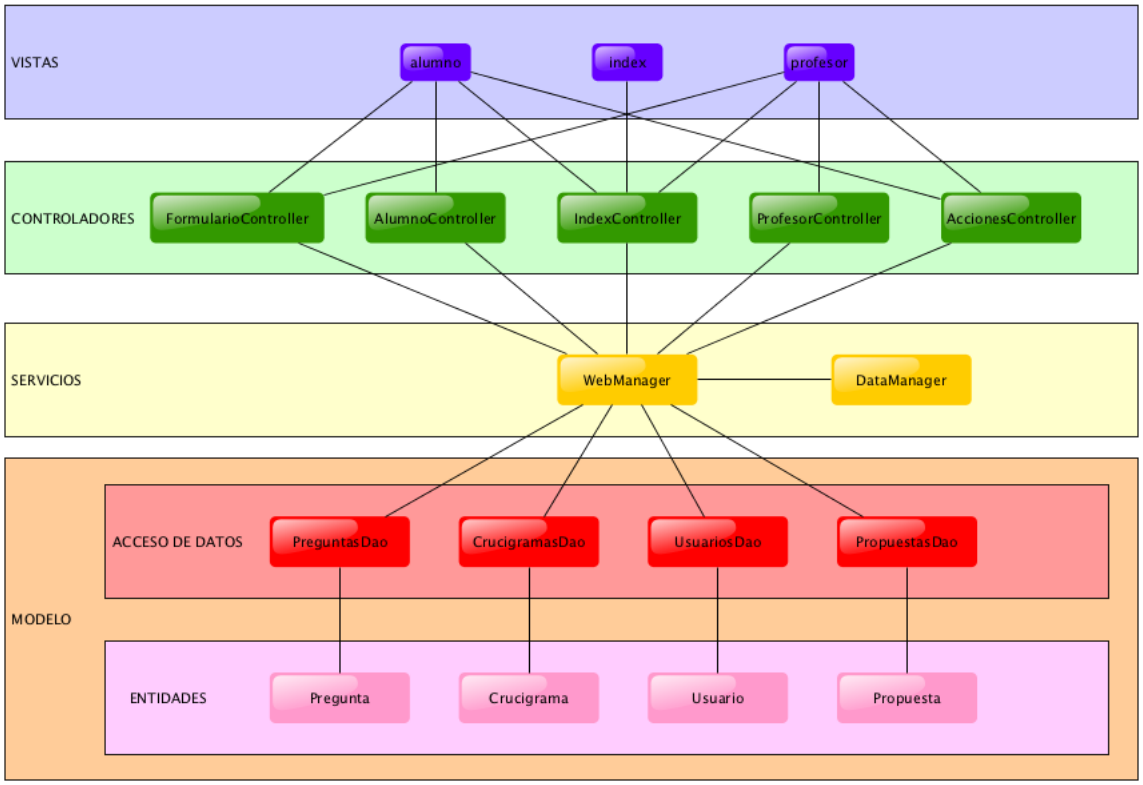


Figura 30: Capas de la aplicación

3.4. Funcionalidad del alumno

Ha llegado el momento de explicar en detalle la funcionalidad concreta de la aplicación así como algunos de los detalles más relevantes en su desarrollo. Comenzaremos con la funcionalidad para los alumnos ya que la aplicación presenta una funcionalidad diferente para alumnos y profesores.

Nos vamos a apoyar en un diagrama de casos de usos para ir definiendo cada uno de los casos más en detalle. En la figura 31 se observa el diagrama con las funcionalidades que se habían marcado como objetivo en la sección 3.1 de definición de tareas.

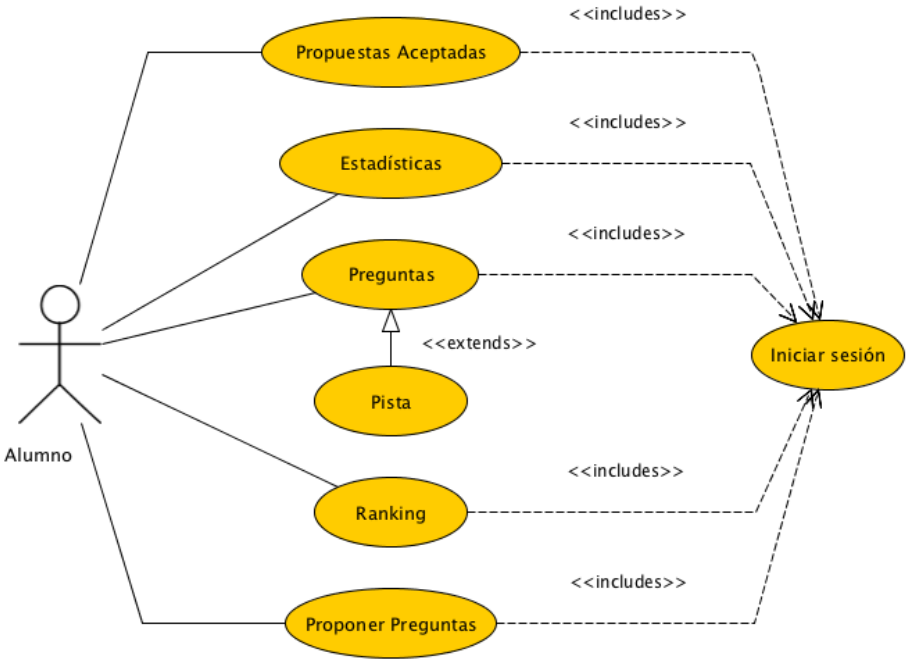


Figura 31: Casos de uso para el alumno

3.4.1. Inicio de sesión

La primera página de OpenWeb será la misma para ambos tipos de usuarios, se trata de la página para iniciar sesión. A partir de aquí el sistema detectará si se trata de un alumno o un profesor gracias a los datos proporcionados, y llevará a cada usuario a su correspondiente sesión de OpenWeb.

El controlador encargado del formulario que aparece en esta vista es *IndexController*. Dependiendo de los datos introducidos *IndexController* mandará un modelo y una vista, si el NIA es de un alumno mandará la vista *alumno* con su modelo correspondiente, si es un profesor la vista será *profesor* y si por el contrario hubo algún tipo de error al introducir los datos, se volverá a mostrar la vista *index* con un mensaje indicando el tipo de problema ocurrido.

Figura 32: Inicio de sesión en OpenWeb

3.4.2. Propuestas aceptadas

Una vez iniciada la sesión se abre la vista *alumno* mostrando un mensaje de bienvenida y el número de propuestas aceptadas por el profesor. Todas las funcionalidades del alumno se muestran en la misma vista, utilizando JSP se puede hacer dinámica dependiendo del modelo de datos obtenido del controlador. En la figura 33 se aprecia el estado inicial de la vista *alumno*.

Tanto la vista como los datos han sido mandados por el controlador *IndexController* al procesar que se trata de un alumno el que está iniciando la sesión. En el modelo se incluye el nombre del alumno para el mensaje de bienvenida y el número de propuestas aceptadas. Pero *IndexController* no es el único controlador que manda este conjunto vista-modelo, existe otro que también permite a la aplicación llegar hasta este mismo punto. Ocurre cuando el usuario pulsa el botón *Inicio*, el controlador *AlumnosController* es el encargado de capturar esta acción y devolver la vista *alumno* a su estado inicial, es decir, al mensaje de bienvenida y el número de propuestas aceptadas.

3.4.3. Estadísticas de los crucigramas resueltos

Esta función muestra una tabla con la nota media que se ha obtenido en los crucigramas y el número de intentos divididos por tema y dificultad. Para utilizar esta función el usuario debe pulsar el botón *Estadisticas* que se encuentra dentro

Figura 33: Página inicial de OpenWeb para alumnos

de la caja *Busqueda*. También hay dos desplegables para poder filtrar por tema, por dificultad, por ambas o por ninguna ya que esta función permite no filtrar. En la figura 34 se puede observar el resultado de una petición de estadísticas filtrando por tema.

El controlador encargado de manejar esta petición, y la gran mayoría de las peticiones que se hacen desde la vista *alumno*, es *AlumnosController*. A través de la capa de servicios se obtienen los datos requeridos y se pasan como un modelo a la vista *alumno* que dibuja de manera dinámica esta información. En el anexo B se pueden ver todos los métodos implementados por la clase *WebManager*. Para esta funcionalidad en concreto se utilizaría el método *obtenerEstadisticas*.

Con esta función el alumno puede hacer un seguimiento de los crucigramas que ha resuelto hasta el momento y las notas obtenidas, motivándole a repetir los crucigramas para mejorar su nota media.

Figura 34: Estadísticas de un alumno filtrando por tema

3.4.4. Preguntas que aparecen en los crucigramas

Con esta función el usuario puede ver que preguntas le han salido en los crucigramas eligiendo un tema y una dificultad. Para poder explicar mejor la información obtenida gracias a esta funcionalidad vamos a apoyarnos en la figura 35. La tabla contiene las siguientes columnas:

- Numero (#): Es el número de la pregunta que se corresponde a su identificador dentro de la base de datos.
- Pregunta: Enunciado de la pregunta.
- Nota media: La nota media entre todas las puntuaciones obtenidas en esa pregunta durante la resolución de los crucigramas.
- Maxima nota: Es la mejor nota obtenida en esa pregunta durante la resolución de los crucigramas.
- Minima nota: La peor nota obtenida.
- Intentos: Número de veces que la pregunta ha sido resuelta en un crucigrama.

- Dificultad: Se corresponde a la dificultad de la pregunta.
- Pista: Un enlace que lleva a una pista acerca de la pregunta.

PREGUNTAS DE LUCIA PAYO

#	PREGUNTA	NOTA MEDIA	MAX. NOTA	MIN. NOTA	INTENTOS	DIFICULTAD	PISTA
2	El mejor amigo del hombre	10.0	10	10	1	Facil	PISTA
3	Capital de España	10.0	10	10	2	Facil	PISTA
4	Herramienta del trabajo del pintor	10.0	10	10	1	Facil	PISTA
5	Utensilio utilizado para comer	10.0	10	10	2	Facil	PISTA
6	Capital de Francia	10.0	10	10	2	Medio	PISTA
7	Capital de Italia	10.0	10	10	2	Medio	PISTA
9	Plataforma 3D parecida a Second Life	10.0	10	10	2	Dificil	PISTA
10	Capital de Alemania	10.0	10	10	2	Medio	PISTA
11	Capital de Bulgaria	10.0	10	10	1	Dificil	PISTA
12	Apellido del actor protagonista del talento de Mr.Ripley	10.0	10	10	2	Dificil	PISTA
13	Apellido del primer presidente de EEUU	4.0	4	4	2	Dificil	PISTA
15	Inventor de la bombilla	10.0	10	10	1	Dificil	PISTA

Figura 35: Preguntas de los crucigramas del tema 2 y dificultad media

En el caso de que el usuario no elija tema y dificultad se le mostrará el siguiente mensaje: Debe elegir un tema y una dificultad.

El controlador para esta función también es *AlumnosController* y funciona de una manera muy similar a la explicada en *Estadisticas*, para el controlador es transparente la clase de información que lleve el modelo, tan solo debe pasárselo a la vista para que esta lo represente. En el caso de *Preguntas* y *Estadisticas* las diferencias se encuentran a partir de la capa de servicios, donde se obtiene la información y se procesa para mandarla al controlador. En concreto se llamaría al método *obtenerPreguntas* de la clase *WebManager* tal y como se explica en el anexo B.

3.4.5. Pistas

Esta funcionalidad está estrechamente ligada con la de obtener preguntas (3.4.4) ya que solo es accesible desde la tabla mostrada. Se trata de un enlace para cada una de las preguntas de la tabla que al pulsarlo te lleva hasta la pista si existe. Es

una función simple de desarrollar ya que solo hay que consultar el campo pista de la entidad palabra determinada.

Esta función pretende ayudar al alumno a encontrar el refuerzo que necesita para estudiar la asignatura. Si el alumno tiene problemas en resolver alguna de las palabras, aquí podrá encontrar cierta información que le puede ser de utilidad.

Hay que destacar que el controlador encargado de manejar esta función es *AccionesController* porque se trata de una acción que se realiza dentro de una tabla y no de un botón. Este controlador se encarga de recibir la petición de pista y de mostrar de nuevo la vista *alumnos* con la información requerida a través de su modelo correspondiente.

3.4.6. Ranking

Se muestran las mejores puntuaciones de los crucigramas filtrando por tema, dificultad o por ambas. En la figura 36 podemos ver un ejemplo de una tabla de ranking para el tema 1 y dificultad difícil. Solo hay cuatro alumnos porque en el sistema de prueba solo se han realizado cuatro crucigramas con esas características, pero la tabla de ranking puede mostrar hasta diez puntuaciones.

RANKING				
#	NOMBRE	TEMA	DIFICULTAD	NOTA
1	Pedro Rodriguez	1	Dificil	100
2	Pedro Rodriguez	1	Dificil	80
3	Lucia Payo	1	Dificil	40
4	Lucia Payo	1	Dificil	30

Figura 36: Ranking para el tema 1 y dificultad difícil

El alumno puede encontrar motivación en intentar superar las puntuaciones de sus compañeros y estar bien posicionado en los rankings. El profesor puede utilizar la información del ranking para ofrecer algún tipo de ventaja a los alumnos en las primeras posiciones.

3.4.7. Proponer preguntas al profesor

El alumno puede proponer sus propias preguntas al profesor, para ello tiene que rellenar un formulario con la pregunta propuesta, la respuesta y el tema al que pertenecería. Si el profesor decide aceptar la propuesta, subirá al marcador de propuestas aceptadas mostradas en el inicio de la aplicación. En la figura 37 está el formulario mencionado.

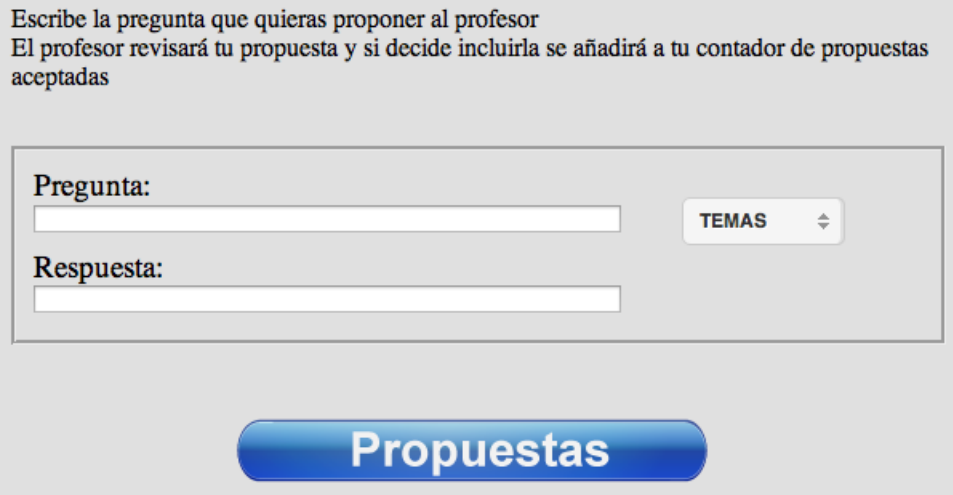


Figura 37: Formulario para mandar una propuesta al profesor

El profesor puede utilizar las propuestas como incentivo para los alumnos, por ejemplo ofreciendo puntos extra a los que hayan conseguido más propuestas aceptadas. Por su parte los alumnos también pueden utilizar las propuestas para escalar en el ranking ya que pueden sugerir preguntas lo suficientemente difíciles para que sus compañeros no puedan resolverlas en los crucigramas. Con esto se consigue una mayor implicación de los alumnos con el temario de la asignatura.

Esta funcionalidad, al tratarse de un formulario, se controla a través de *FormularioController*, que es el encargado de reunir los datos introducidos por el alumno y pasárselos a *WebManager* para que a través de la capa de acceso a datos, cree una nueva entrada en la tabla *propuestas*. En respuesta devuelve la misma vista, es decir *alumnos*, que muestra un mensaje de confirmación: Se ha mandado correctamente tu propuesta al profesor.

3.5. Funcionalidad del profesor

Algunas de las funciones del profesor son muy similares a las de los alumnos como vamos a ver a continuación. Al igual que con las funcionalidades para los alumnos, se presenta un diagrama de casos de usos en la figura 38.

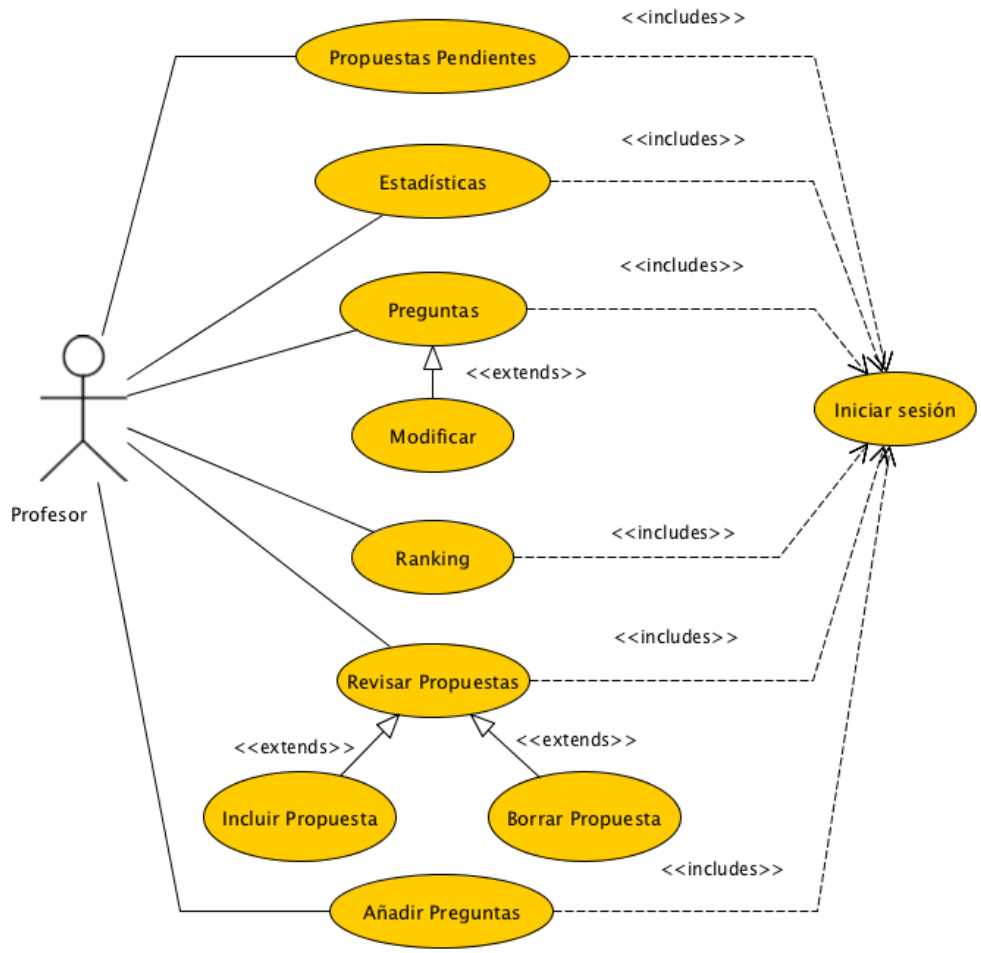


Figura 38: Casos de uso para un profesor

El inicio de sesión es el mismo que para los alumnos al igual que la funcionalidad de ranking, así que nos saltaremos estos dos puntos. Se puede consultar el inicio de sesión en la sección 3.4.1 y el ranking en la sección 3.4.6. Para iniciar sesión como profesor, al no disponer éste de NIA, se ha fijado el NIA 1000 para su identificación.

3.5.1. Propuestas pendientes de revisión

Al igual que ocurría con los alumnos, cuando el profesor inicia sesión le aparece una página de inicio con un mensaje de bienvenida y, en este caso, el número de propuestas que tiene pendientes de revisión. La vista encargada de mostrar la aplicación es *profesor* que del mismo modo que *alumno* se encarga de mostrar cada una de las funcionalidades porque es dinámica gracias a JSP. La figura 39 se corresponde con la página inicial para el profesor.

Figura 39: Página inicial de OpenWeb para el profesor

El controlador *IndexController* es el encargado de mandar la vista *profesor* cuando procesa que se trata de un profesor el usuario que está iniciando la sesión. Manda el modelo con las propuestas pendientes de revisión, que se corresponde con

el número de entradas que haya en ese momento en la tabla *propuestas*. Al igual que en la vista *alumnos*, existe un botón llamado *Inicio* que se maneja a través del controlador *ProfesorController* y que devuelve la vista al estado inicial, mostrando el mensaje de bienvenida y el número de propuestas pendientes de revisión.

3.5.2. Estadísticas de un alumno

El profesor puede obtener la misma información que obtienen los alumnos con la funcionalidad de estadísticas. Para ello se incluye un desplegable con los nombres y los NIAs de todos los alumnos registrados en la aplicación, el profesor puede elegir a cualquiera de ellos y consultar su progreso. La información obtenida es la misma que la que obtiene el alumno (sección 3.4.3). La figura 40 muestra un ejemplo con las estadísticas del alumno Pedro Rodriguez para todos los temas y dificultades (sin filtrado). Aunque en la aplicación hay disponibles hasta ocho temas, para las pruebas solo hay dos plenamente funcionales, es por ello que en la tabla de la figura 40 solo salgan dos temas.

Figura 40: Estadísticas de un alumno para todos los temas y dificultades

Aunque el controlador encargado de realizar esta función es *ProfesorController*, el procesado es el mismo que en el caso de los alumnos, es decir, que a partir de la capa de controladores se han realizado las mismas operaciones. Tanto *Alumnos-Controller* como *ProfesorController* hacen la misma petición a la capa de servicios

(*WebManager*), obteniendo el mismo modelo pero mostrándolo en una vista u otra según sea el caso.

Gracias a esta funcionalidad el profesor puede hacer un seguimiento de los alumnos y la utilización de los crucigramas.

3.5.3. Añadir preguntas nuevas

El profesor puede añadir las preguntas que saldrán en los crucigramas de Opensim gracias a esta funcionalidad. A través de un formulario podrá definir toda la información necesaria para el correcto funcionamiento del sistema. Unas instrucciones le serán mostradas para que tenga en cuenta ciertos aspectos, pueden apreciarse en la figura 41. Si se incumple cualquiera de las reglas se mostrará un mensaje de error.

Al ser un formulario está controlado por *FormularioController*. Reúne toda la información y si es correcta se la pasa a la función *incluirPregunta* (Anexo B) para que a través de la capa de acceso a datos se almacene como una nueva entrada. En caso de ser incorrecta la información en el formulario, el controlador le indicará a la vista el mensaje de error correspondiente.

3.5.4. Preguntas que aparecen en los crucigramas

Esta funcionalidad, aunque en apariencia pueda parecer la misma que para los alumnos, es diferente. También se muestran las preguntas que han salido filtrando por tema y dificultad del crucigrama, pero los datos mostrados cambian por otros de mayor interés para el profesor. En la figura 42 se muestra la tabla de preguntas para crucigramas del tema uno y dificultad difícil. A continuación se da una breve descripción de cada uno de los campos mostrados:

- Numero (#): Es el número de la pregunta que se corresponde a su identificador dentro de la tabla.
- Pregunta: Enunciado de la pregunta.
- Nota media: La nota media entre todas las puntuaciones obtenidas por todos los alumnos al resolver la pregunta.
- Desv. estándar: Se trata de la desviación estándar de las notas.

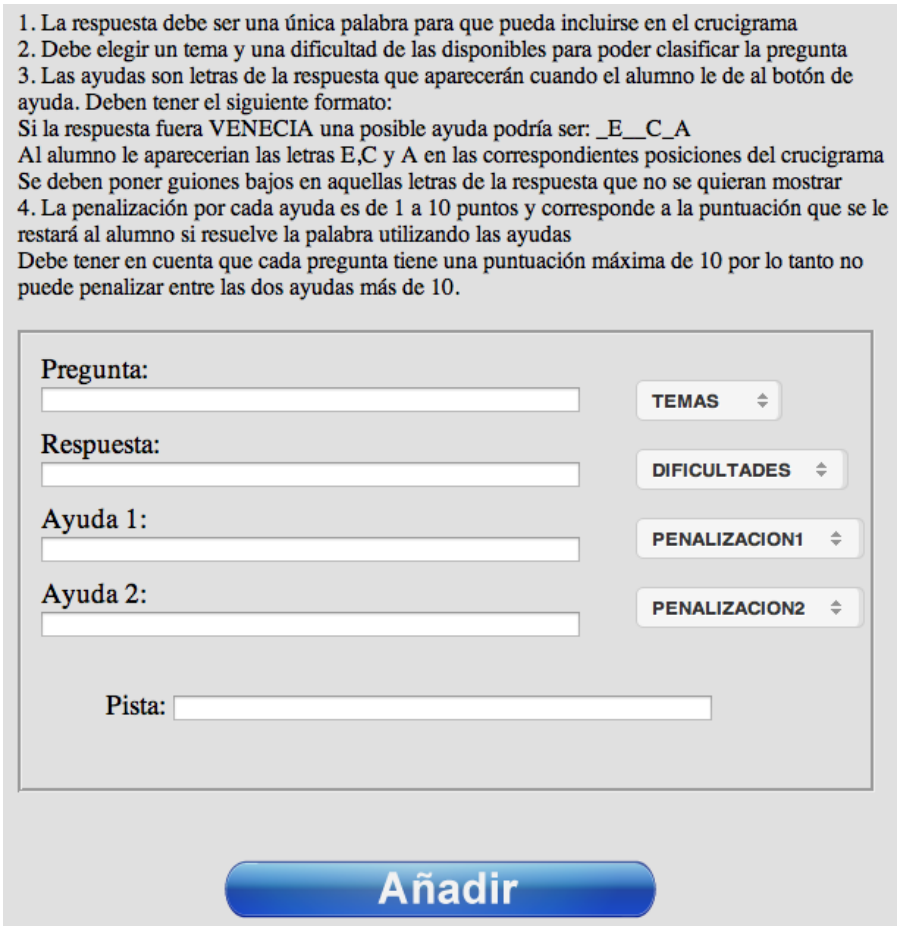
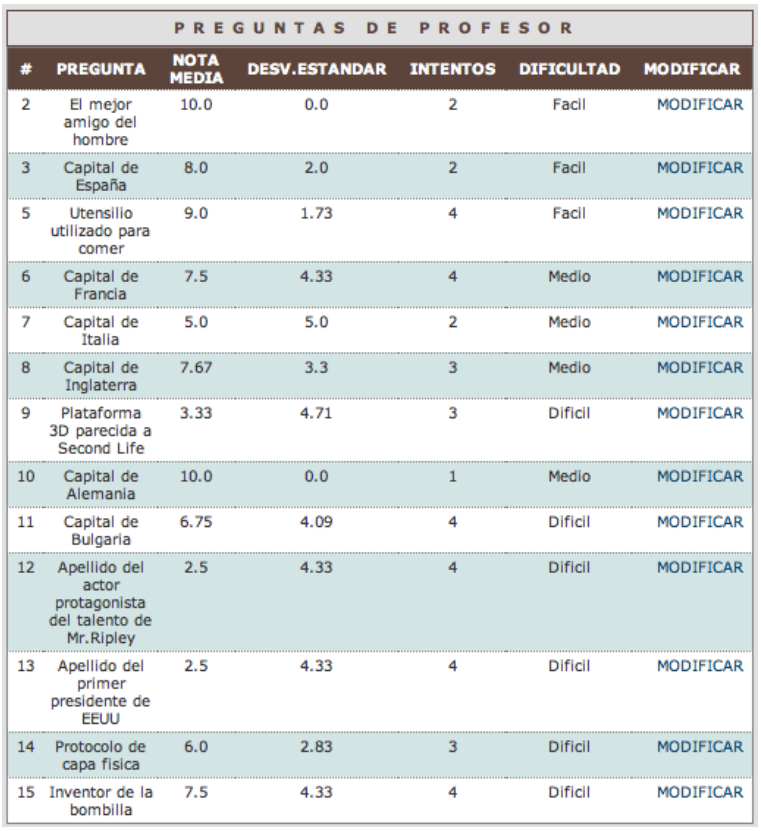


Figura 41: Añadir preguntas nuevas

- Intentos: Número de veces que la pregunta ha sido resuelta en un crucigrama por todos los alumnos.
- Dificultad: Se corresponde a la dificultad de la pregunta.
- Modificar: Un enlace para modificar la pregunta.

Al igual que ocurría con los alumnos, el profesor debe elegir un tema y una dificultad para poder visualizar las preguntas, en caso contrario aparecerá el siguiente mensaje: Debe elegir un tema y una dificultad.

Con esta información el profesor puede hacer un seguimiento del número de veces que aparece una pregunta en los crucigramas, pero lo que realmente se pretendía con esta función es que el profesor pueda ver que preguntas están dando



PREGUNTAS DE PROFESOR

#	PREGUNTA	NOTA MEDIA	DESV.ESTANDAR	INTENTOS	DIFICULTAD	MODIFICAR
2	El mejor amigo del hombre	10.0	0.0	2	Facil	MODIFICAR
3	Capital de España	8.0	2.0	2	Facil	MODIFICAR
5	Utensilio utilizado para comer	9.0	1.73	4	Facil	MODIFICAR
6	Capital de Francia	7.5	4.33	4	Medio	MODIFICAR
7	Capital de Italia	5.0	5.0	2	Medio	MODIFICAR
8	Capital de Inglaterra	7.67	3.3	3	Medio	MODIFICAR
9	Plataforma 3D parecida a Second Life	3.33	4.71	3	Dificil	MODIFICAR
10	Capital de Alemania	10.0	0.0	1	Medio	MODIFICAR
11	Capital de Bulgaria	6.75	4.09	4	Dificil	MODIFICAR
12	Apellido del actor protagonista del talento de Mr.Ripley	2.5	4.33	4	Dificil	MODIFICAR
13	Apellido del primer presidente de EEUU	2.5	4.33	4	Dificil	MODIFICAR
14	Protocolo de capa fisica	6.0	2.83	3	Dificil	MODIFICAR
15	Inventor de la bombilla	7.5	4.33	4	Dificil	MODIFICAR

Figura 42: Preguntas de los crucigramas del tema uno y dificultad difícil

más problemas a los alumnos y si les ha puesto la dificultad adecuada. La nota media no es del todo fiable por si sola, ya que es importante también saber si las notas de los alumnos están muy espaciadas entre sí. La desviación estándar da información acerca de lo dispersas que están las notas de los alumnos, siendo 0.0 la menor distancia entre notas, es decir, todos los alumnos sacaron la misma nota, y 5.0 la mayor distancia, es decir, que los alumnos sacaron o un cero o un diez en la pregunta. Recordemos que la nota de una pregunta en concreto se veía afectada por el número de pistas que pedía el alumno al resolverla, por lo que un alumno solo puede sacar cuatro notas concretas en la pregunta: cero porque no la resolvió o la resolvió incorrectamente, diez menos la penalización de haber pedido dos pistas, diez menos la penalización de haber pedido una pista o diez porque la resolvió sin ayuda. Con toda esta información el profesor puede decidir si alguna de las preguntas necesita modificación, por ejemplo si una pregunta está catalogada como fácil pero la nota media es baja y la desviación típica también es pequeña, quizás deba elevarse la dificultad a medio o difícil.

3.5.5. Modificar las preguntas

Esta funcionalidad es accesible solo a través de la anterior (3.5.4). En la tabla de preguntas, la última columna se corresponde con un enlace para modificar la entrada correspondiente. Al pinchar sobre él la aplicación nos muestra el mismo formulario para añadir con los datos de la pregunta a modificar tal y como se muestra en la figura 43.

Como pasaba con la funcionalidad pista del alumno explicada en la sección 3.4.5, la función modificar se encuentra dentro de una tabla por lo tanto el controlador encargado de su funcionamiento es *AccionesController*. De la misma manera que para pista, se encarga de recibir la petición de modificar la pregunta correspondiente mostrando después el formulario con los datos de la pregunta. Sin embargo, el controlador encargado de capturar las modificaciones de la pregunta es *FormularioController* ya que es el mismo formulario que utiliza la función añadir y se gestiona de la misma manera. En este caso el controlador *FormularioController* llama a la función *modificarPregunta* (Anexo B) pasándole el identificador de la pregunta a modificar además de la información. Resumiendo, *AccionesController* se encarga de mostrar el formulario con los datos de la pregunta que se va a modificar y *FormularioController* se encarga de persistir la modificación a través de la capa de servicios.

3.5.6. Revisar las propuestas

Como ya se ha explicado en la sección 3.4.7, el alumno puede proponer preguntas. Las propuestas aparecerán en una tabla tal y como se muestra en la figura 44. Se indica quién fue el alumno que propuso la pregunta y se tiene la opción de incluirla o de eliminarla.

3.5.7. Incluir propuesta

Si el profesor decide incluir la propuesta del alumno, se abrirá un formulario como el de la figura 41 con los campos pregunta, respuesta y tema rellenos. El profesor debe completar el resto de campos, estimando la dificultad, y modificando, si lo cree conveniente, cualquiera de los datos proporcionados por el alumno. Al igual que pasaba con la funcionalidad de modificar preguntas de la sección 3.5.5, a esta funcionalidad se accede a través de un link en una tabla, es decir que el controlador

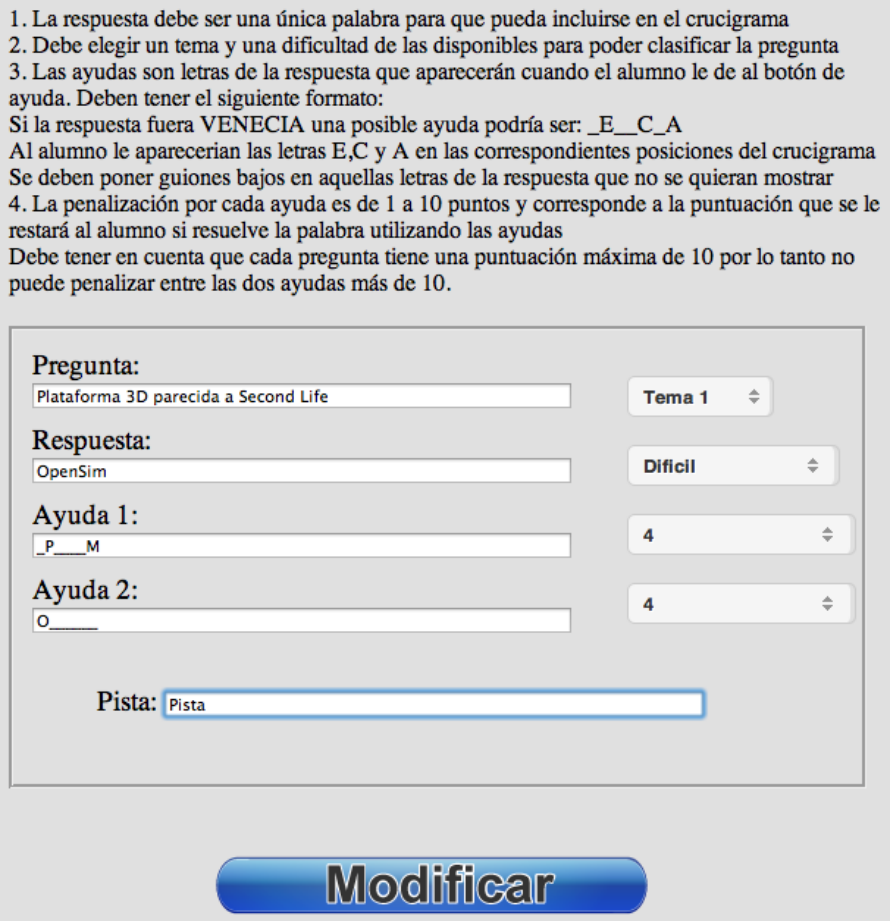
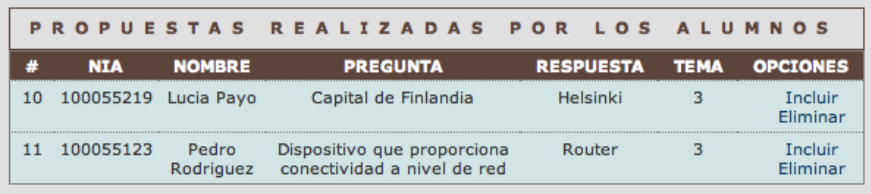


Figura 43: Formulario para modificar una pregunta

encargado de abrir el formulario con los campos correspondientes completados es *AccionesController*, pero a la hora de reunir los datos cuando el profesor pulsa el botón para añadir la nueva pregunta es el controlador *FormularioController* el que se encarga del proceso. Las operaciones que se realizan son muy similares a las realizadas cuando se añade una pregunta nueva, la única diferencia es que hay que sumarle una propuesta aceptada al alumno.

3.5.8. Eliminar propuesta

Si por el contrario el profesor piensa que la pregunta no debe de estar entre las posibles en los crucigramas elegirá la opción de eliminar. La implementación



PROPUESTAS REALIZADAS POR LOS ALUMNOS

#	NIA	NOMBRE	PREGUNTA	RESPUESTA	TEMA	OPCIONES
10	100055219	Lucia Payo	Capital de Finlandia	Helsinki	3	Incluir Eliminar
11	100055123	Pedro Rodriguez	Dispositivo que proporciona conectividad a nivel de red	Router	3	Incluir Eliminar

Figura 44: Tabla con las propuestas hechas por los alumnos

de esta funcionalidad es muy sencilla, el controlador *AccionesControler* recibirá la petición de eliminar la propuesta y llamará al método *eliminarPropuesta* de *WebManager* (Anexo B). Una vez eliminada no volverá a aparecer entre las propuestas pendientes de revisión.

4. Conclusiones y trabajos futuros

4.1. Conclusiones

La enseñanza y el aprendizaje cada vez abren más sus espectros a nuevas posibilidades, en el caso de este proyecto se ha explotado la posibilidad de utilizar un mundo virtual 3D y una plataforma web como apoyo a la educación. La idea de utilizar mundos virtuales en la enseñanza ya ha sido explotada por multitud de proyectos alrededor del mundo por lo que es un terreno que está en plena expansión y desarrollo.

La aplicación desarrollada engloba dos componentes o ambientes de desarrollo diferentes: por un lado tenemos una interfaz 3D, que viene dada por el mundo virtual OpenSim, donde los alumnos pueden elegir el crucigrama a resolver. Les ayuda a repasar sus conocimientos de una manera diferente y más atractiva a la habitual. Por otro lado, tenemos la herramienta OpenWeb que permite manipular y recoger información de los crucigramas. Se trata de una interfaz web 2D que se plantea como un servicio web clásico donde los alumnos pueden ver su avance y encontrar información útil para el estudio. Para el profesor OpenWeb constituye una herramienta útil que le brinda la posibilidad de motivar a sus alumnos y controlar el entorno educativo.

En lo que respecta a los crucigramas en OpenSim se han conseguido alcanzar los requisitos inicialmente marcados como funciones básicas que debían estar presentes. Los alumnos tienen libertad para resolver el crucigrama cuando quieran, pueden centrarse en temas o dificultades concretas atendiendo más a sus necesidades, se trata de un entorno visual e intuitivo que resulta agradable y vistoso.

En la parte de OpenWeb se han desarrollado dos interfaces diferentes, cada una enfocada al tipo de usuario que la va a utilizar. La interfaz del profesor tiene un enfoque de administración y seguimiento de los alumnos, con herramientas para gestionar las preguntas. Sin embargo la interfaz de los alumnos está más enfocada a hacer un seguimiento personal y como herramienta de apoyo al estudio, además de motivar a los usuarios para seguir resolviendo crucigramas.

Para la realización del sistema completo han sido necesarios varios módulos de diferente naturaleza además de establecer comunicación entre ellos. Por un lado el servidor de OpenSim debía comunicarse con el exterior para poder generar el crucigrama y persistir los datos obtenidos de los alumnos, con este objetivo se

desarrollaron varios programas PHP alojados en un servidor Apache. Por otro lado se debían persistir todos los datos y tener acceso a ellos desde ambas aplicaciones, el crucigrama en OpenSim y la plataforma OpenWeb. Para esta tarea se eligió la base de datos MySQL, compartida por ambas partes y que representa el núcleo central del sistema completo. OpenWeb fue desarrollado en Java por lo que necesita de un servidor de aplicaciones para funcionar, como por ejemplo Tomcat.

El sistema propuesto en el presente proyecto puede utilizarse como complemento a un sistema de educación tradicional, siendo un elemento más en el conjunto de herramientas utilizadas por el profesor para el desarrollo de su asignatura.

4.2. Trabajos futuros

Sería interesante probar el sistema actual en un entorno real, por ejemplo con un grupo reducido de alumnos, replicando el crucigrama de OpenSim tantas veces como fuera necesario para que todos los alumnos pudieran tener oportunidad de resolver uno. Después de analizar el funcionamiento de las herramientas en un entorno aislado se podrían incluir o mejorar los siguientes puntos:

- Crear crucigramas colaborativos. Actualmente cada crucigrama puede ser resuelto tan solo por un único estudiante, en el futuro sería interesante que los alumnos pudieran colaborar unos con otros en la resolución de los crucigramas.
- Crear una herramienta de registro para que un profesor pueda dar de alta su asignatura en el sistema y los alumnos puedan registrarse para esa asignatura en concreto. En la versión actual debe ser un administrador el que incluya a los usuarios en el sistema y no existe una diferenciación por asignaturas.
- Mejorar visualmente OpenWeb, haciéndola más atractiva e intuitiva para los usuarios.
- Mejorar visualmente los crucigramas en OpenSim utilizando herramientas como Maya.
- Integrar la herramienta OpenWeb con la página web de la universidad, teniendo acceso a notas reales en la asignatura o a aula global.
- Desarrollo de otros objetos educativos lúdicos dentro del mundo virtual. La idea se puede extender a otros objetos diferentes, no solo a crucigramas, por ejemplo se podría hacer un juego de preguntas y respuestas donde hay que elegir entre varias opciones.

- Incluir nuevos idiomas en OpenWeb.
- Llevar la aplicación a un entorno masivo con muchos usuarios, teniendo en cuenta factores como la eficiencia, capacidad del servidor, ancho de banda, etc.

5. Presupuesto y tiempo dedicado

Se han analizado dos escenarios distintos para el cálculo del presupuesto asociado a este proyecto: por un lado, la tarea realizada hasta ahora, es decir, la fase de desarrollo; por el otro, al ser ésta una aplicación ideada para ser utilizada en un entorno educativo, se ha considerado interesante estimar cuál sería el presupuesto necesario para su funcionamiento.

5.1. Escenario de desarrollo

El presupuesto asociado a este escenario se desglosa en los siguientes costes:

5.1.1. Costes de personal

Para llevar a cabo este cálculo es necesario realizar un análisis de las distintas tareas en que se divide este trabajo y estimar la duración de cada una de ellas:

Fases	**Dedicación en meses**	**Coste(Euros)**
Estudio de las tecnologías para los crucigramas	1	5388,78
Diseño de la aplicación crucigramas	2	5388,78
Implementación de la aplicación crucigramas	2	5388,78
Estudio de las tecnologías para OpenWeb	1	2694,39
Diseño de OpenWeb	2	5388,78
Implementación de OpenWeb	3	8083,17
Escritura de la memoria y posterior revisión	2	5388,78
TOTAL	13	35027,07

Los datos considerados han sido 21 días laborables por cada mes y jornada de 8 horas diarias. Todos los roles necesarios para el desarrollo del proyecto (diseño, programación en distintos lenguajes, etc.) fueron asumidos por la autora. El salario considerado ha sido el establecido en las plantillas de la universidad para un Ingeniero, esto es, 2694,39 por hombre y mes.

Aunque la estimación se ha hecho distinguiendo claramente las fases por simplicidad, en realidad se fueron introduciendo cambios en el diseño también en la fase de implementación al comprobar de forma práctica ciertas mejoras o límites. De igual modo, la estimación en meses es orientativa, ya que no sólo se ha tra-

bajado en días laborables y también durante algunas fases del proyecto no se ha desarrollado la jornada completa, por lo que su duración real ha sido mayor.

Al total obtenido hay que realizarle un ajuste, teniendo en cuenta que el ingeniero realizador del proyecto no está aún en posesión del título, por lo que se le puede aplicar un factor del 70 % en el cálculo. Con esto, el coste de personal asciende a 24518,95 euros.

5.1.2. Costes de material

En lo que a hardware se refiere se ha necesitado de un ordenador portátil personal valorado en 1100 euros y una máquina que alojara los diferentes servidores necesarios valorada en 400 euros. No ha habido costes de software ya que todas las tecnologías utilizadas se trataban de software libre.

5.1.3. Costes indirectos

Se calculan como el 20 % del coste total y engloban todos aquellos gastos como electricidad, conexión a Internet, etc. En este proyecto ascienden a 5203,79.

5.1.4. Total

El presupuesto total para el desarrollo del sistema es de 31222,74

5.2. Escenario de producción

Se contempla el uso inicial de la aplicación durante un cuatrimestre y con un grupo de 20 alumnos. Teniendo en cuenta este escenario, el presupuesto destinado para la utilización de la herramienta se puede desglosar en:

5.2.1. Costes de personal

Contemplan la existencia de un ingeniero encargado del mantenimiento y administración de los servidores. Para el cálculo se considera una dedicación de 1 hora semanal, lo cual implica, aproximadamente, 4 horas al mes y, por tanto, 16 horas al cuatrimestre.

Por otra parte, hay que considerar la instalación de los servidores OpenSim, MySQL, Apache y Tomcat y la puesta a punto para su funcionamiento. Esta tarea puede llevar dos días con jornada de 8 horas, lo cual supone un total de 16 horas.

Contando para ambas tareas el salario de un Ingeniero, estimado a partir de los datos del apartado anterior, el presupuesto total para el personal ascendería a 135 euros.

5.2.2. Costes de material

Al igual que en el apartado anterior, sólo es necesario tener en cuenta los gastos de hardware.

En primer lugar, es necesario un servidor que soporte las cuatro tecnologías mencionadas en el apartado anterior y que dé servicio a unos 20 clientes. Una máquina de estas características tiene un coste de aproximadamente 1500 euros.

En segundo lugar, se consideran 20 estaciones de trabajo para los alumnos, las cuales pueden ser equipos de 500 euros, resultando así un total de 10000 euros.

5.2.3. Costes indirectos

Se calculan como el 20 % del coste total y engloban todos aquellos gastos como electricidad, conexión a Internet, etc. En este caso ascienden a 2327 euros.

5.2.4. Total

El presupuesto total para el escenario de producción seria de 13962 euros.

5.3. Tiempo dedicado

Por último y para finalizar el presente proyecto fin de carrera se presenta un diagrama de Gantt que comprende un periodo desde Junio del 2010 hasta Octubre de 2012.

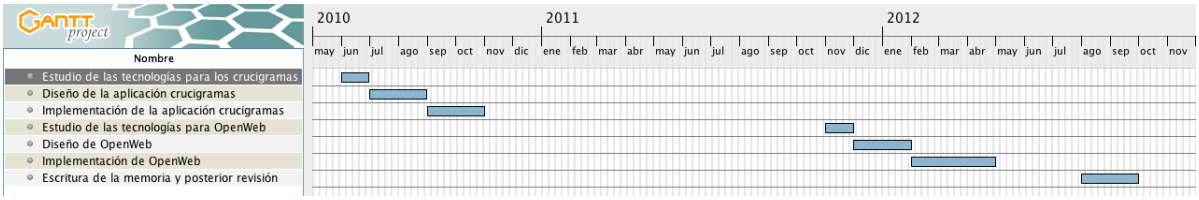


Figura 45: Diagrama de Gantt

Referencias

[1] "Aprendizaje electrónico," Junio 2012. http://es.wikipedia.org/wiki/Aprendizaje_electronico.

[2] J. Cabero, "Bases pedagógicas del e-learning," Abril 2006. http://www.uoc.edu/rusc/3/1/dt/esp/cabero.pdf.

[3] J. M. Calés and R. Hurtado, "Tomografía del e-learning en las universidades europeas," Junio 2012. http://www.elearningamericalatina.com/radiografias/rad_9.php.

[4] "Opensim." http://opensimulator.org/wiki/Main_Page.

[5] "¿qué es second life?." http://secondlife.com/whatis/?lang=en-US.

[6] "Linden script language." http://wiki.secondlife.com/wiki/LSL_Portal/es.

[7] "Backtracking." http://commons.wikimedia.org/wiki/File:Branch%26bound_low.jpg?uselang=es.

[8] "Opensimulator wiki," Marzo 2012. http://opensimulator.org/wiki.

[9] "Modos de funcionamiento de opensim." http://opensimulator.org/wiki/Configuration.

[10] "What is mysql?," 2012. http://dev.mysql.com/doc/refman/5.1/en/what-is-mysql.html.

[11] "Maven," Julio 2012. http://es.wikipedia.org/wiki/Maven.

[12] "Introducción a maven," Noviembre 2011. http://www.genbetadev.com/java-j2ee/introduccion-a-maven-ii-project-object-model.

[13] "Patrón mvc," Enero 2012. http://www.rincondeloajeno.com/php-y-la-arquitectura-mvc/.

[14] "¿qué es la inyección de dependencias?," Abril 2011. http://www.genbetadev.com/paradigmas-de-programacion/que-es-la-inyeccion-de-dependencias.

[15] "How spring mvc module works," Agosto 2010. http://www.roseindia.net/spring/how-spring-mvc-works.shtml.

[16] "Spring mvc: Web mvc framenwork." http://static.springsource.org/spring/docs/2.5.x/reference/mvc.html.

[17] "Introducción a jpa." http://www.davidmarco.es/blog/entrada.php?id=144.

[18] "Jsp tutorial." http://www.jsptut.com/Getfamiliar.jsp.

Anexos

A. Instalación de los componentes

En este anexo se explica como instalar todos los elementos necesarios para el funcionamiento de la aplicación.

Junto con la presente memoria se adjuntan los programas y archivos necesarios para el funcionamiento del sistema, entre ellos se encuentra el servidor de OpenSim. Se encuentra comprimido en un archivo llamado *opensim-0.7-rc2.zip* que contiene el conjunto de objetos que forman un crucigrama y dos usuarios. Simplemente hay que descomprimir el archivo en la máquina que se vaya a usar como servidor y tener en cuenta que OpenSim utiliza los puertos 9000 y 9001 en este caso.

A la hora de arrancar el servidor OpenSim hay que tener en cuenta si el sistema operativo donde se está trabajando es Windows o Unix. Dentro de la carpeta *bin* se encuentra un archivo llamado *OpenSim.exe*, si nos encontramos en Windows tan solo hay que pinchar dos veces sobre él de la manera habitual y si nos encontramos en un sistema Unix necesitaremos un programa adicional llamado *mono* para poderlo arrancar. Este programa puede descargarse e instalarse desde los repositorios utilizando la siguiente línea: *sudo apt-get install mono-complete*. Una vez instalado ya será posible arrancar el servidor de OpenSim, desde una consola nos colocamos dentro de la carpeta *bin* y escribimos: *sudo mono OpenSim.exe*.

OpenSim abrirá su propia consola de comandos desde donde se puede gestionar el servidor. Un comando útil es añadir usuarios. Nos vamos a apoyar en la figura 46 para explicar su funcionamiento. Al escribir *create user* en la consola de OpenSim el sistema nos preguntará por los datos necesarios para la creación del usuario, nombre, apellido, contraseña y e-mail. Es interesante matizar que para que el sistema funcione, los usuarios deben tener el mismo nombre en el servidor OpenSim y en la tabla *usuarios* de la base de datos.

Existen otros comandos interesantes en la consola de OpenSim, para verlos todos junto con una breve descripción tan solo hay que escribir *help*.

Para poder conectarse como usuario a una región de OpenSim se necesita un cliente. En la red hay muchos clientes diferentes que servirían para el propósito

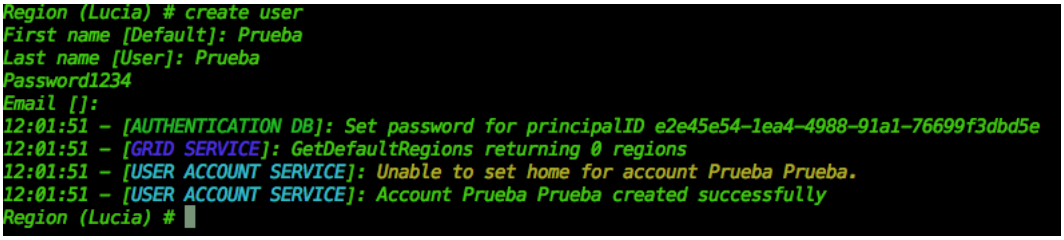


Figura 46: Comando para crear nuevos usuarios en OpenSim

aquí marcado, por ejemplo podemos utilizar Imprudence [11] que tiene versión para Linux, Windows y Mac.

Lo primero que debemos hacer es configurar el servidor al que nos queremos conectar. Para ello hay un botón en el cliente, *Grid Manager*, al pulsarlo se abre una ventana de configuración. A la izquierda hay un listado con todos los grids configurados por defecto en el cliente, para añadir uno nuevo pulsamos el botón *Add New Grid*. Al hacerlo, el formulario de la derecha se quedará vacío para que podamos introducir los datos de nuestro servidor, tan solo es necesario rellenar los campos *Grid Name*, será el nombre con el que se identificará nuestro grid, y *Login URI*, que se corresponde con la dirección IP seguida del puerto. A continuación se pulsaría el botón *Apply* y se cerraría la ventana *Grid Manager*. Existe un desplegable con los grids configurados, debemos elegir el que acabamos de crear e introducir nombre, apellido y contraseña. Si todos los datos han sido correctamente introducidos, al pulsar el botón *Iniciar sesión* el cliente debería mostrar nuestro avatar en la región creada con el crucigrama. La figura 47 representa lo explicado en este párrafo.

Otro de los elementos que hay que configurar es la base de datos MySQL. En las pruebas realizadas teníamos el servidor MySQL en la misma máquina que los servidores Apache y Tomcat, por lo que en la configuración de la base de datos *localhost* es la URL indicada. Si no fuera de esta manera, y la base de datos estuviera en otra máquina diferente, habría que cambiar esta configuración. En el caso de OpenWeb habría que modificar el fichero *persistente.xml* alojado en la carpeta *src/main/resources*. Para los archivos *.php* alojados en el servidor Apache, habría que cambiar la configuración de la base de datos de los ficheros *bd_info*, *crucigrama.class.php* y *evaluacion.php* todos dentro de la carpeta *lib*. A parte de la URL hay que cambiar la configuración de usuario y contraseña.

[11] Se puede descargar de: http://wiki.kokuaviewer.org/wiki/Downloads

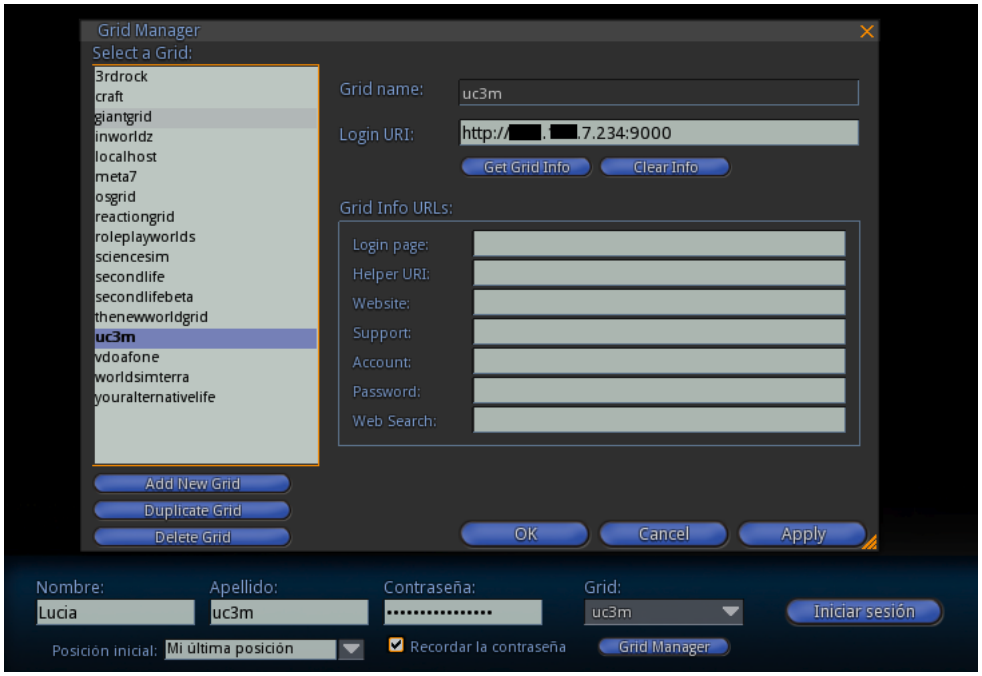


Figura 47: Ventana de configuración del cliente

Para generar las tablas y el conjunto de datos utilizados en las pruebas se proporciona el fichero *proyecto.sql*. Para volcarlo hay que crear primero una base de datos llamada *crucigrama* y después escribir la siguiente línea de código: *mysql -u username -p -h localhost crucigrama <proyecto.sql*. Una vez hecho esto estarían listas todas las tablas necesarias con los datos de prueba.

A continuación habría que instalar un servidor Apache para alojar los programas *.php* encargados de la generación y recogida de datos del crucigrama. Estos archivos se encuentran en la carpeta llamada *crucigrama* que se incluye junto con la presente memoria. OpenSim se comunica directamente con estos programas por lo que habría que modificar parte del script del objeto *Crucigrama* para que enviara las peticiones al servidor adecuado.

Por último hay que desplegar la aplicación OpenWeb en el servidor Tomcat. Se incluye todo el código fuente dentro de la carpeta llamada *openweb* y el archivo desplegable *openweb-1.0.war* se encuentra dentro de la carpeta target.

También se incluyen los scripts que contienen los objetos de Opensim en la carpeta *Script* y las texturas que se han necesitado incluir en la carpeta *Texturas*.

B. WebManager

La clase *WebManager.java* es la encargada de realizar toda la lógica de negocio de la aplicación OpenWeb y por ello es la clase con más peso. A continuación se va a dar una pequeña descripción de los métodos implementados en esta clase ya que pueden ayudar a la comprensión del proyecto.

En la figura 48 se encuentran los métodos más relevantes y sus atributos. Todos los atributos que terminan por *Dao* son los objetos encargados del manejo de entidades, estos objetos no se instancian en la clase *WebManager* sino que son inyectados gracias al framework Spring MVC. Aunque en la figura 48 no aparecen, existen métodos set para todos estos objetos a través de los cuales se realiza la inyección.

```
public class WebManager implements Constantes{

        private UsuariosDao usuariosDao;
        private CrucigramasDao crucigramasDao;
        private PreguntasDao preguntasDao;
        private PropuestasDao propuestasDao;
        private DataManager dataManager;

        public int validarUsuario(int nia, String password) throws FileNotFoundException, IOException{...}

        public String nombreUsuario(int nia){...}

        public Map<Integer,Object> obtenerEstadisticas(int tem, int dif, int nia){...}

        public Map<Integer, Map<String,Object>> obtenerPreguntas(int tema, int dificultad, int nia){...}

        public Map <Integer,Map<String,Object>> ranking(int tema, int dificultad){...}

        public Map<Integer,String> buscarAlumnos(){...}

        public int numeroPropuestas(){...}

        public Map<Integer,Map<String,Object>> propuestas(){...}

        public void incluirPregunta(String pregunta, String respuesta, String tema, String dificultad,
                String ayuda1, String ayuda2, String pen1, String pen2, String pista){...}

        public  Map<String,Object> detallesPregunta(int numero){...}

        public void modificarPregunta(int id,String pregunta, String respuesta, String tema,
                String dificultad, String ayuda1, String ayuda2, String pen1, String pen2, String pista){...}

        public Map<String,Object> detallesPropuesta(int num){...}

        public void incluirPropuesta(int id,String nia, String pregunta, String respuesta, String tema,
                String dificultad, String ayuda1, String ayuda2, String pen1, String pen2, String pista){...}

        public void eliminarPropuesta(int id){...}

        public int propuestasAceptadas(int nia){...}

        public String pista(int num){...}

        public void proponer(String tema, String pregunta, String respuesta, int nia){...}
```

Figura 48: Clase WebManager

validarUsuario: Método para comprobar si la contraseña es correcta para el nia pasado en el inicio de sesión. Devuelve una de las constantes definidas en la clase *Constantes.java*:

- NIA_NO_ENCONTRADO: En caso de que el nia pasado no exista en la base de datos.
- OK_PROFESOR: Si la pareja nia/contraseña es correcta y se trata de un profesor.
- OK_ALUMNO: Si la pareja nia/contraseña es correcta y se trata de un alumno.
- PASSWORD_INVALIDA: Si la contraseña introducida no es correcta.

nombreUsuario: Método sencillo para obtener el nombre de usuario a partir del nia pasado.

obtenerEstadisticas: En este método se reúnen todos los datos necesarios para mostrar las estadísticas. Se le pasa el tema, la dificultad y el nia del alumno y devuelve un objeto de tipo *Map* que relaciona claves con valores. En este objeto se encuentra mapeada toda la información que hay que mostrar en la vista.

obtenerPreguntas: Similar al método *obtenerEstadisticas* solo que reuniendo la información sobre las preguntas. También hay que pasarle el tema, la dificultad y el nia del alumno y devuelve un objeto de tipo *Map*.

ranking: Devuelve en un objeto de tipo *Map* con el ranking en el tema y dificultad pasados como parámetros.

buscarAlumnos: Obtiene todos los alumnos registrados en el sistema y los devuelve dentro de un objeto tipo *Map*.

numeroPropuestas: Método que devuelve el número de propuestas pendientes de revisión por parte del profesor.

propuestas: En este método se reúnen las propuestas hechas por los alumnos y se devuelven dentro de un objeto de tipo *Map*.

incluirPregunta: Este método se utiliza para incluir una nueva pregunta a las ya existentes en la base de datos. Los parámetros pasados son los necesarios para poder crear una entidad *pregunta*.

detallesPregunta: Pasando el identificador de la pregunta devuelve el valor de todos sus atributos en un objeto *Map*.

modificarPregunta: Se le pasa el identificador único de la pregunta a modificar y los nuevos valores de los atributos.

detallesPropuesta: Pasando el identificador de la propuesta devuelve el valor de todos sus atributos *Map*.

incluirPropuesta: Método muy parecido a *incluirPregunta* con la única diferencia de que se pasan también el identificador único de la propuesta para eliminarla de las propuestas pendientes, y el nia del alumno para incluirla en sus propuestas aceptadas por el profesor. La propuesta será incluida como una nueva pregunta a la base de datos.

eliminarPropuesta: Elimina la propuesta de las propuestas pendientes.

propuestasAceptadas: Pasando el nia del alumno devuelve el número de propuestas hechas por este alumno que ha aceptado el profesor.

pista: Como parámetro se pasa el identificador único de una pregunta y devuelve la pista para esa pregunta.

proponer: Crea una propuesta nueva y la incluye en propuestas pendientes de revisión por parte del profesor.